VŠB – Technická univerzita Ostrava

Fakulta elektroniky a informatiky

Fotbal LEGO robotů

LEGO Robot Soccer

2013 Bc. Achiles Savidis

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Achiles Savidis**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Fotbal LEGO robotů
LEGO Robot Soccer

Zásady pro vypracování:

Cílem práce je navrhnout a realizovat architekturu hry fotbalu robotů, založenou na LEGO robotech. Výsledkem práce by měla být funkční hra LEGO robotů, využívající mimo jiné i dynamický kamerový systém. Student naváže na práci předchůdce, jež obsahuje návrh konstrukce LEGO robotů a analýzu obrazu nutnou pro tvorbu kamerového systému.

1. Seznámení se s aktuálním stavem hry fotbalu LEGO robotů.
2. Vytvoření dynamického kamerového systému a návrh značení robotů.
3. Návrh kompletní architektury hry. Zde je možno využít stávající řešení strategií a taktik aktuálně používaných v simulátoru, případně přijít s vlastním návrhem.
4. Návrh a provedení úloh pro otestování funkčnosti kompletní hry.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Václav Svatoň**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 19.7.2013 Savčák .....

Zde bych rád poděkoval vedoucímu své diplomové práce Ing. Václavu Svatoňovi za pomoc a podporu, kterou mi poskytl v průběhu tvorby této diplomové práce. Dále také RNDr. Elišce Ochodkové, Ph.D. nejen za rady k daným úlohám, ale také za pomoc s dalšími organizačními záležitostmi této diplomové práce.

Abstrakt

Fotbal robotů je již řadu let oblíbenou disciplínou vysokoškolských studentů se zaměřením na robotiku a umělou inteligenci. Jak časem dochází k technologickým pokrokům, začínají být pro širokou veřejnost dostupné technologie a možnosti, které dříve nebyly. Jednou takovou možností je LEGO MINDSTORMS NXT.

Cílem této práce je přejmout stávající části, kterými jsou: roboti postavení z LEGO MINDSTORMS NXT, softwarová analýza obrazu, strategie a rozhraní pro ovládání robotů přes Bluetooth. Následně zrevidovat tyto dílčí projekty a zakomponovat v jednu aplikaci, která bude schopna hrát Fotbal LEGO robotů.

Klíčová slova

LEGO MINDSTORMS NXT, fotbal LEGO robotů, robotický fotbal, roboti, analýza obrazu, OpenCv, detekce objektů v obraze, ovládání prostřednictvím Bluetooth

Abstract

Robot soccer is for many years favorite scientific field of many university students of robotics or artificial intelligence. As technology progresses, a wide public becomes able to access different technologies than ever before. One of possibilities like these is LEGO MINDSTORMS NXT.

Goal of this work is to work with already existing parts such as: robots build using LEGO MINDSTORMS NXT, software video analysis, strategy core and interface for remote control of the robots via Bluetooth. Subsequently make revision of these minor projects and connect it into one application able to play LEGO robot soccer.

Keywords

LEGO MINDSTORMS NXT, LEGO robot soccer, robot soccer, robots, video analysis, OpenCv, object detection in image, Bluetooth remote control

Seznam použitých zkratek a symbol

C++/CLI	- C++ Common Language Infrastructure
CLR	- Common Language Runtime
COM	- COMmunication port
GUI	- Grafické uživatelské rozhraní
LED	- Light-Emitting Diode
Bgr	- barevný formát (modrá, zelená, červená)
Hsv	- barevný formát (odstín, sytost, jas)
Rgb	- barevný formát (červená, zelená, modrá)

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

1.1 Úvod do problematiky

Fotbal LEGO robotů, jak již název napovídá, je hlavním námětem této práce. Robotika však nezahrnuje pouze roboty, kteří jsou schopni podílet se na nějakém sportu, ale zahrnuje mnoho oblastí a má svou vlastní historii, která sahá daleko do minulosti. Stačí se jen podívat na wikipedii na Historii robotů [1] nebo na přehled oblastí robotiky či typů robotů [2] a narazíte na vyčerpávající výčet.

S postupným zdokonalováním technologií, ať už hardwarových nebo softwarových, se roboti stále více podobají lidem a stále více jsou schopni vykonávat činnosti, které jinak musely být vykonávány lidmi. Může se ale robot plně podobat člověku?

Výrazný pokrok ve vývoji robota, který by se měl chovat a vypadat jako člověk udělala firma Honda se svým robotem *ASIMO* [3], která se snaží lidem dokázat, že robot-humanoid je budoucností tohoto oboru. Před tím než však mohou humanoidní roboti vykonávat nějakou činnost, je dobré vyřešit problémy jednotlivých úkonů pomocí robotů přímo specializovaných pro danou úlohu.

Pro některé činnosti jsou navíc humanoidní roboti naprosto nevyhovující. Můžeme se podívat třeba na situaci auto, které se řídí samo. Existence humanoidního robota by byla naprosto nadbytečná. Jaký typ konstrukce robota jsme se ale rozhodli použít při realizaci *fotbalu LEGO robotů*? A z čeho vlastně samotná myšlenka *fotbalu LEGO robotů* vznikla?

Podle provozovatele turnaje *RoboCup*[4], byla myšlenka vytvořit roboty, kteří by hráli fotbal, byla poprvé zmíněna profesorem Alanem Mackwothem v roce 1992 v práci nazvané "*On Seeing Robots*". Během *International Joint Conference on Artificial Intelligence* (Mezinárodní konference o umělé inteligenci) v Kanadském Montrelu konané v srupnu roku 1995 bylo poprvé ohlášeno vytvoření *TheFirst Robot World Cup Soccer Games and*

Conferences. Po provedení plánovaných příprav byl první oficiální turnaj robotického fotbalu, nazvaný *RoboCup*, uspořádán v roce 1997, kdy se zúčastnilo více než 40 týmů.

RoboCupSoccer (fotbal robotů) se hraje s využitím malých robotů, kteří jsou identičtí pro každý tým. Neboli, týmy nemají možnost vytvořit si vlastního robota, který by se nějakým parametrem odlišoval od ostatních. Jakou návaznost na tento turnaj má tedy fotbal LEGO robotů?

Fotbal LEGO robotů se snaží určitým způsobem navázat na *RoboCupSoccer* v tom, že základní logická část zůstala stejná. Hra se stále hraje ve dvou týmech, kdy každý tým má určitý počet hráčů. Počet hráčů se může měnit, logickým předpokladem však je, že oba týmy budou mít v jedné hře shodný počet hráčů. Na protilehlých kratších stranách hřiště jsou umístěny dvě brány, kdy každý tým má právě jednu bránu. Navíc je na hřišti umístěn míč. Každý tým se snaží za pomocí robotů dostat míč do brány druhého týmu. Celá hrací plocha je snímána kamerou, která je umístěna přímo nad hřištěm. Obraz z kamery je analyzován počítačem, jsou vyhodnoceny pozice jednotlivých objektů na hřišti a podle toho systém sám vyhodnotí a rozhodne kam se má který robot jeho týmu posunout. Ovládání každého týmu je tedy plně automatické a člověk do ovládání během hry nijak nezasahuje, každý pouze předem vytvoří strategie, kterými se budou roboti během hry řídit. Tyto části jsou stejné jak u *RoboCupSoccer*, tak i u *fotbalu LEGO robotů*. Co se tedy liší od původního návrhu *RoboCupSoccer*?

Rozdílným prvkem mezi *fotbalem LEGO robotů* a *RoboCupSoccer* je to, že u *fotbalu LEGO robotů* se počítá s tím, že roboti budou poskládáni ze stavebnice *LEGO MINDSTORMS NXT*. Další důležitou změnou, která se rozchází s původní myšlenkou *RoboCupSoccer* je snaha sestavit systém tak, aby bylo možné hru snímat kamerou ze strany. Je snahou, aby při přípravě hřiště nebylo potřeba stavět kolem něj velkou konstrukci. Stačí malý rám po jedné straně hřiště nebo správně umístěný stativ, tak aby z dostatečně velkého úhlu dokázala kamera zabrat celé hřiště. *Analýza obrazu* by samozřejmě s velkou pravděpodobností nefungovala správně, kdyby kamera nedokázala zachytit celé hřiště

LEGO MINDSTORMS NXT je stavebnice, která byla vytvořena společností *LEGO*. Na rozdíl od dalších stavebnic této formy i jiných firem, *LEGO MINDSTORMS NXT* obsahuje navíc ještě řídící jednotku, několik motorků a sadu senzorů. Díky tomu, že stavebnice je prodávána za relativně příznivou cenu, může si dnes robota sestavit téměř každý. Nejen tedy již dříve zmíněného robota typu humanoid, ale také vozidla či jiné struktury, kterou je uživatel schopen ze stavebnice poskládat. V tomto projektu bude tedy cílem za pomocí těchto robotů sestavených z *LEGO MINSTORMS NXT* zrealizovat verzi fotbalu robotů nazvanou *fotbal LEGO robotů*.

1.2 Struktura dokumentu

Základní myšlenky týkající se hry a robotů, kteří ji hrají, již byly představeny, je tedy na čase podívat se, jakým způsobem bude rozdělena tato práce, tak aby byly postupně předloženy všechny relevantní informace, které se jí týkají.

Nejprve ze všeho bude představen projekt ve stavu, v jakém byl přejat od předchůdců podle jednotlivých oddílů, do kterých spadají. Každá část bude samostatně popsána. Následuje část věnovaná konstrukci robota-fotbalisty a dojde na popis toho, z čeho je vytvořen – již představenou stavebnici *LEGO MINSTORMS NXT*. Část textu bude věnována tomu, z jakých nejdůležitějších stavebních bloků se skládá a jaký je rozdíl mezi verzí *NXT 1.0*,*NXT 2.0* a nově uvedenou verzí *EV3*.

V této části bude také představen a vysvětlen původní způsob signalizace robotů pomocí *LED diod* a způsob, kterým byl nahrazen. Bude ukázáno, že některé principy původního značení byly zachovány, bylo však potřeba odstranit některé negativní efekty. Zde přijde na řadu i krátké zvážení jiných metod značení.

Další významnou částí tohoto projektu, které bude věnován prostor, je *analýza obrazu*. Zde bude popsáno, jaké technologie byly použity při vývoji výsledné aplikace. Také jaké operace je třeba s obrázky provést, aby bylo možné je jednoduše zpracovat a jaké postupy byly použity při detekci míče a hráčů.

Následující významnou částí je *modul strategií*. Nad touto částí projektu byla provedena řada drobných změn, většina však byla stejného charakteru, proto je ji věnováno v této práci méně prostoru. Přesto nebude zapomenuto na popis uživatelsky vytvořených souborů pravidel, které *modul strategií* využívá při svých výpočtech.

Dále bude věnována pozornost na třetí hlavní část projektu, která je zaměřena na ovládání robotů fotbalistů, neboli *Bluetooth rozhraní*. Existuje několik projektů zaměřených na ovládání robotů postavených z *LEGO MINDSTORMS* v obou vydaných verzích, a tak náplní této kapitoly bude krátký rozbor jednotlivých zvažovaných knihoven a stručné zdůvodnění závěrečné volby. Dále také krátký popis jaké funkce *Bluetooth rozhraní* vykonává.

Na závěr teoretické části bude věnován prostor grafickému rozhraní aplikace a popisu některých jejích funkcí, které mohou být užitečné při nastavování hry *fotbalu LEGO robotů*. Stejně jako v ostatních částech, i zde budou připojeny obrázky, které mají za úkol demonstrovat, jak vytvořená aplikace vypadá a funguje.

Po uzavření teoretické části, bude věnována část textu třem hlavním praktickým experimentům, kde bude popsáno, jak probíhalo provádění experimentů, a jak byli roboti a aplikace schopni splnit cíle projektu v reálném nasazení.

2 Počáteční stav projektu

Hlavní náplní této práce bylo vzít již vytvořené části a spojit je do jednoho funkčního celku, případně vylepšit jejich funkcionalitu. Student, který se na tuto problematiku zaměřoval v předchozí fázi, vypracoval diplomovou práci se stejným názvem: „*Fotbal LEGO robotů*“ a připravil cestu dalšímu rozvoji, když se ve své práci zaměřil na tři prvky: konstrukci „robota fotbalisty“, analýzu obrazu a ovládání robotů pomocí *Bluetooth*. Každá z těchto částí však byla samostatným prvkem a výsledkem tedy nebyla jedna aplikace, která by měla za úkol automaticky ovládat roboty tak, aby hráli fotbal.

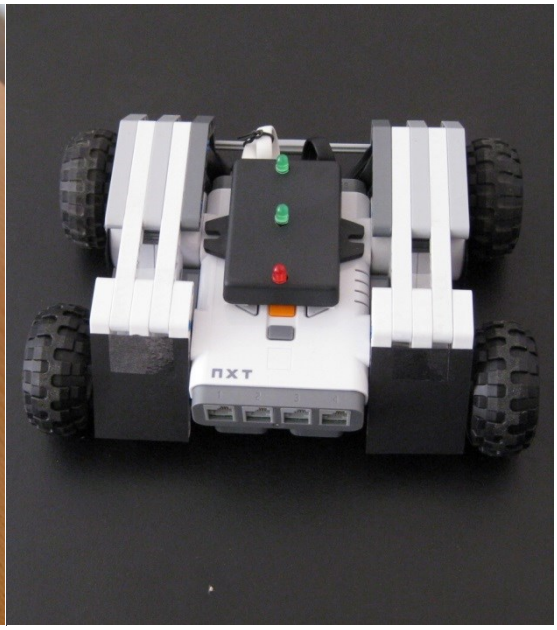
Dále jsem od vedoucího mé diplomové práce obdržel projekt, který obsahoval simulátor hry fotbalu robotů. Kromě vizuální části, aplikace obsahovala také modul, který sloužil k rozhodování kam se má který robot posunout, aby docházelo k pohybu robotů podle jistých pravidel. Tato část, která v této práci hrála také významnou roli, je dále v textu nazývána *modul strategií* a tak se na ni i budu dále odkazováno.

Cílem mé práce bylo spojit všechny tyto samostatné prvky do jednoho celku tak, aby vznik dynamický systém. Tento systém by mělo stačit před spuštěním nakonfigurovat pomocí pár jednoduchých kroků. Při spuštění by systém měl být schopen řídit hru *LEGO* robotů.

2.1 Konstrukce robotů

Jednou z částí, kterou řešil Zdeněk Neustupa, bylo to, že navrhnul konstrukci robota, která nejen oproti jiným návrhům byla stabilnější a tak ne hrozí převrácení robota při náhodné kolizi s mantinelem nebo jiným robotem, ale jeho konstrukce také zajišťuje, že vrchní plocha robota není zkosená a tudíž na něj lze velmi snadno umístit panel s identifikačními led diodami. Tím, že panel s *LED diodami* není nijak zkosený, jsou všechny *LED diody* v jedné rovině a jsou tím vyřešena jistá úskalí, která by mohla působit problémy.

Obrázek 1: Jeden z původních návrhů konstrukce robota, Zdroj[5]

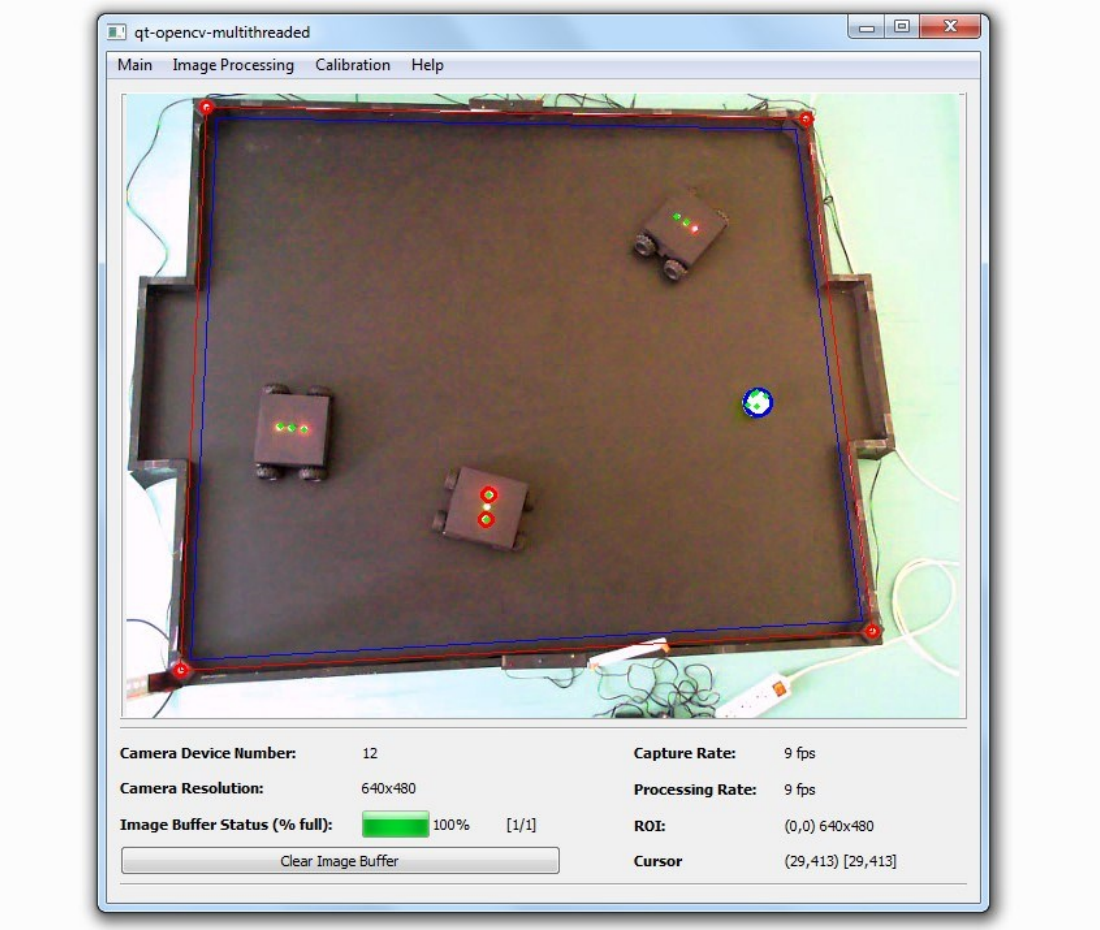


Obrázek 2: Konstrukce robota, v počáteční fázi projektu, Zdroj[6]

Více pozornosti na aspekt signalizačního značení a další body však bude zaměřen v následujících sekcích, které obsahují bližší popis softwarové aplikace a následně konstrukce robota a identifikačního panelu.

2.2 Analýza obrazu

Analýza obrazu je asi nejvýraznější softwarová část tohoto projektu, při přirovnání k lidskému tělu představuje jakoby jeho oči. Tato část byla částečně přejata od předchůdce Zdeňka Neustupy. Tato aplikace byla napsána v programovacím jazyku C++ a využívala knihovny *Qt* a *OpenCv*. Byla postavena na projektu *qt-opencv-multithreaded*[7], který vytvořil Nick D'Ademoa publikoval jej pod *MIT Licencí*[8]. Dále můj předchůdce do projektu zakomponoval vlastní prvky, které z aplikace s obecným zaměřením udělaly aplikaci zaměřenou na fotbal lego robotů.



Obrázek 3: Původní aplikace sloužící k analýze obrazu, Zdroj: vlastní

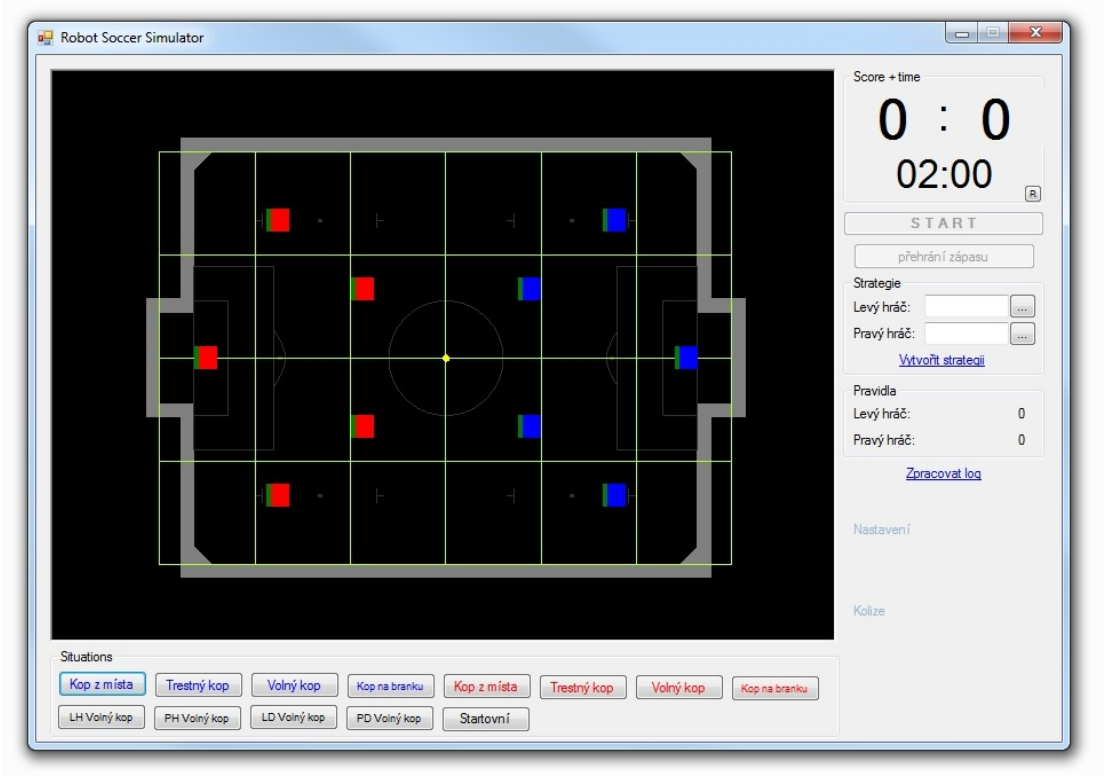
Jak je na obrázku č. 3 vidět, aplikace je schopna na první pohled v jednoduchém *GUI* zobrazit hřiště a zvýraznit diody, nalezených robotů. Tato aplikace ale v hlavním menu ukrývá celou řadu možných nastavení a kalibrace, které ji dělají mírně těžkopádnou.

Praktický test ukázal, že je velmi náchylná na osvětlení a úspěšnost nelezení robota není nijak velká. Navíc sice dokáže rozdělit nalezené podezřelé body do skupin a tím určit, které paří společnému robotovi, nedokáže ale identifikovat barvu jednotlivých *LED diod* a proto ani identifikovat jednotlivé hráče.

2.3 Strategie

Strategie je softwarová část projektu, která obsahuje kódy, které rozhodují o všech pohybech robotů týmu, kterému přísluší. Je to vlastně mozek celé hry. Tento *modul strategií* příjme data od části programu, který zajišťuje analýzu obrazu, získaná data zpracuje a navrhne kam se má který robot posunout.

Modul strategií byl v počáteční fázi projektu součástí simulátoru hry *fotbalu LEGO robotů*. Z tohoto simulátoru bylo třeba *modul strategií* vyjmout a začlenit jej do vytvářené aplikace. Tato část kódu byla napsána v programovacím jazyku *C#* a je součástí této práce ji začlenit do jednotného systému, ne však upravovat její zdrojový kód mimo úpravy nezbytných k tomu, aby tato část projetu dokázala bez problémů komunikovat s ostatními částmi.



Obrázek 4: Simulátor hry *fotbalu LEGO robotů*, Zdroj vlastní

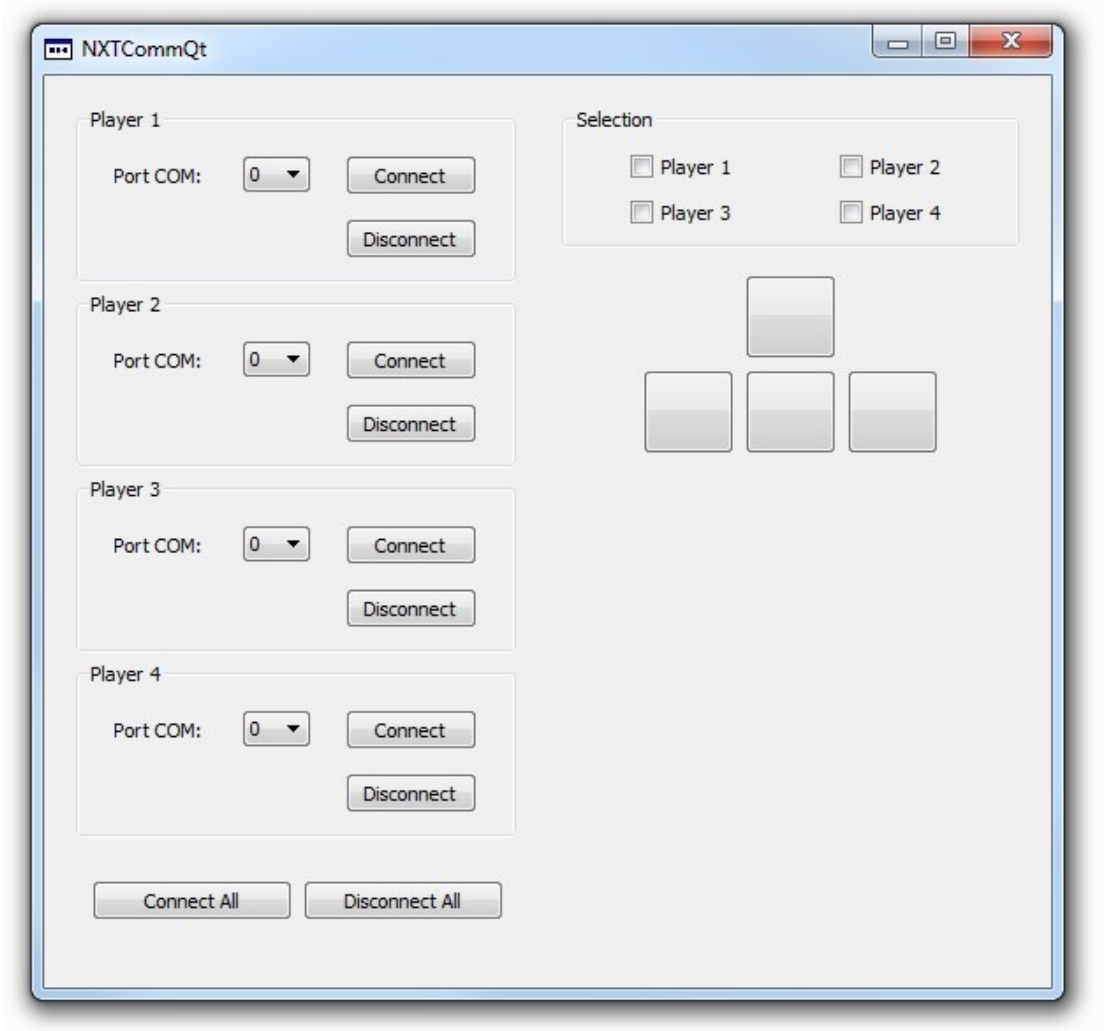
2.4 Ovládání robotů

Roboti jsou ovládání pomocí rozhraní *Bluetooth*. K připojení robotů přes *Bluetooth* je využíváno *COM* portů. Každý robot se k počítači připojuje pod určitým *COM* portem, který je pro každé zařízení v rámci jednoho počítače obvykle jiný. Tudíž lze každého robota, který byl nalezen pomocí analýzy obrazu, také nalézt v síti *Bluetooth* a poslat správnému robotovi pokyny, které mu opravdu náleží.

Druhá aplikace, která byla přejata od předchůdce, je aplikace, která obsahovala knihovny pro ovládání *LEGO MINDSTORMS NXT* robota pomocí *Bluetooth* a formulář s ovládacími tlačítky, který tyto knihovny využíval tak, že po jednoduchém připojení, bylo možné robota

ovládat pomocí šipek a tudíž s ním pohybovat vpřed, vzad, doleva i doprava jak to všichni známe z použití u běžných autíček na dálkové ovládání. Aplikace byla napsána v programovacím jazyku C++.

Tato aplikace již byla navržena a vytvořena, aby ovládala daného robota v tom stavu, jak byl navržen pro hru *fotbal LEGO robotů*. Kromě ovládání vpřed, vzad, doleva, doprava však již neobsahovala žádný přepočet souřadnic.



Obrázek 5: Původní aplikace pro ovládání robotů, Zdroj vlastní

3 Konstrukce robota

Jak bylo v úvodu nastíněno, robot je složený ze stavebnice *LEGO*, která se v několika ohledech odlišuje od jiných běžně dostupných stavebnic. Jak vlastně stavebnice vypadá? CO obsahuje? A jaké technologie, které by mohly být užitečné pro náš projekt, podporuje? O tom budou pojednávat následující kapitoly.

3.1 LEGO MINDSTORMS NXT

Většina robotů v tomto projektu je postavena z *LEGO MINDSTORMS NXT 2.0*, přesto jeden robot byl postaveno z téhož typu stavebnice, ale ve verzi *1.0*. Pokusme se teď rozebrat, jaké součástky robot používá, ale protože se roboti poskládaní ze stavebnic jiných verzí téměř neliší, dojde pouze na popis stavebnici *LEGO MINDSTORMS NXT 2.0*. U této stavebnice jsou součástí základní výbavy stavebnice hlavně dvě součástky, které jsou u robota sestaveného pro účel hry *fotbal LEGO robotů*, a proto jim bude věnováno trochu pozornosti. Je to hlavní řídicí jednotka *NXT IntelligentBrick* (zkráceně *Brick*) a druhou je *servo motor*.

V době psaní této diplomové práce již společnost *LEGO* neprodává *LEGO MINDSTORMS NXT 2.0*. Místo toho ale počátkem roku 2013 oznámila vydání nové verze pod označením *LEGO MINDSTORMS EV3*, které by na trh mělo vyjít na podzim roku 2013. Konec této kapitoly se proto stručně zaměří na rozdíl mezi jednotlivými verzemi.

Obrázek 6: Stavebnice *LEGO MINDSTORMS NXT 2.0*, Zdroj[18]

3.1.1 Řídicí jednotka - Brick

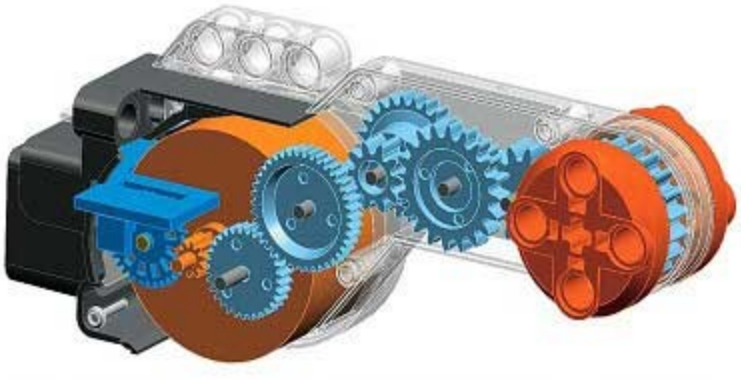
Brick, centrální řídicí jednotka stavebnice má podle svého anglického názvu tvar kostky. Je součástí každého *LEGO* robota fotbalisty a také se očekává, že bude součástí každého robota postaveného z této stavebnice. Slouží k ovládání všech ostatních komponent a robot, který by ji neobsahoval, by se tak stal statickým a ztratil by tak kontrolu nad všemi periferiemi.

Brick obsahuje tři výstupní porty pro zapojení robota, které jsou značeny písmeny v abecedě (*A*, *B*, *C*) a čtyři výstupní porty pro zapojení senzorů, označené čísly (*1*, *2*, *3*, *4*).

Jak je vidět z obrázku 2, porty pro senzory nejsou v našem případě využity. Do portu pro motorky *A* a *C* jsou zapojeny levý a pravý motor. Port *B* je využit pro napájení identifikačního panelu robotů.

3.1.2 Motor

Motorky dodávané ke stavebnici *LEGO MINDSTORMS NXT* jsou servomotory se zabudovaným rotačním čidlem. To se velmi hodí, pokud požadujeme přesnost natočení motoru.



Obrázek 7: Tvar motorku a jeho vnitřní konstrukce, Zdroj[9]

3.1.3 LEGO MINDSTORMS NXT 2.0

Sada *LEGO Mindstorms NXT 2.0* byla vydána 5. Srpna 2009 a obsahuje 619 stavebních kousků. Verze *2.0* je až na pár detailů prakticky identická s verzí *1.0*. Jedním rozdílem je to, že ve verzi *2.0* se řídicí jednotka *Brick* dočkala nové verze firmware, který je ovšem možno nainstalovat i na jednotku *Brick* z předchozí verze, čímž se rozdíl stírá. Další rozdíl mezi těmito verzemi, už má ale vliv na moji diplomovou práci.

Tímto malým a ve většině případů nepodstatným problémem je tvar a velikost koleček, které se nasazují na motorky. Nejen, že to způsobuje, že náš robot fotbalista postavený ze stavebnice ve verzi *1.0* jemně tře o zem a tudíž je jeho pohyb méně efektivní, ale hlavní problém tkví v tom, že vzhledem k tomu, že kolečka v každé verzi mají jiný poloměr, tak pokud bychom přepočítávali pohyb robota na otáčky motorku, dojdeme v každé verzi k jiným výsledkům, a tudíž by se robot ve verzi *1.0* mohl chovat nepředvídaným způsobem, což není žádoucí

3.1.4 LEGO MINDSTORMS EV3

V nové verzi *LEGO MINDSTORMS EV3*, kde *EV* znamená „*evolution*“, se nadšenci pro robotiku mohou dočkat hlavně proměny samotné řídicí jednotky *Brick*. Nejen, že nyní vypadá jinak, ale také dostala vyšší rozlišení displeje a novější procesor (*ARM 9*, oproti předchozí verzi, kde byl *ARM 7*), na kterém nyní běží operační systém *Linux*. Více informací,

o chystané nové verzi této stavebnice společnost *LEGO* publikovala na jejich webových stránkách v sekci *Frequently Asked Questions* pro *LEGO MINDSTORMS EV3*[10].

To byla krátká zmínka o nově chystané verzi a následuje další podkapitola, týkající se konstrukce robota a tedy část o použitém identifikačním značení.

3.2 Identifikační značení

Určení polohy každého hráče na hřišti je samozřejmě nezbytnou součástí celého procesu hraní fotbalu robotů. Vzhledem k tomu, že identifikace počítačem probíhá prostřednictvím analýzy obrazu, je navíc logické, že způsob, jakým zajistíme, aby se podařilo najít přesné souřadnice a natočení každého robota, stejně jako jeho odlišení od ostatních robotů na hřišti, musí být vizuální.

3.2.1 Značení pomocí LED diod

Řešení identifikace robotů, které bylo použito jako součást diplomové práce Zdeňka Neustupy využívá LED diod, které jsou napájeny přímo z výstupního portu robotovy řídicí jednotky.

Pro potřeby tohoto projektu dostatečně slouží modul, sestavený ze tří klasických *LED diod* usazených v řadě za sebou, který je vidět na obrázku 8. Na obrázku 9 je pak demonstrováno, jak toto značení vypadá při samotném použití při zachycení kamerou, která byla při tomto typu značení použita.

Obrázek 8: Panel LED diod pro identifikaci robota, Zdroj vlastní

Obrázek 9: Roboti značeni za pomoci LED diod, Zdroj vlastní

Byla uvažována možnost, kdy by panel byl osazen jiným počtem *LED diod* případně, kdy by nebyly led diody usazené v řadě za sebou, ale použilo by se jiné označení. Takový přístup můj předchůdce považuje za méně výhodný či nepotřebný, než způsob, který je zde použit a v této práci se došlo ke stejnému závěru. Tři *LED diody* zabírají méně místa oproti řešení s větším počtem *LED diod*, takže nemusí být umístěny tolik u kraje a nedochází k takovým chybám detekce robotů, že by systém k sobě spojil *LED diody*, které k robě nepatří. To by bylo možno kompenzovat jiným uskupením *LED diod*, umístění všech *LED diod* v jedné řadě však dále zjednodušuje operace při identifikování informací o jednotlivých robotech.

Osazení a umístění *LED diod* na robotovi bylo konstruováno od počátku tak, že vzdálenost mezi *LED diodami* umístěnými na stejném panelu je mnohem menší než vzdálenost mezi *LED diodami*, které k sobě nepatří. Aplikace tudíž funguje tak, že spojuje do jednotlivých skupin nálezy, které jsou k sobě navzájem nejblíže a zároveň nepřekračují předem stanovenou maximální vzdálenost. Díky tomu již nebylo třeba provádět žádný matematický výpočet, který by operaci komplikoval a zpomaloval.

Nízký počet použitých Led diod se může nejdříve zdát jako nedostatečný, obzvláště když je nutné vyhradit první slot pouze pro červenou barvu, aby bylo možné rozpoznat natočení robota, což zároveň znamená, že v posledním slotu červená barva být použita nesmí. Při použití tří barev LED diod jsme ale schopni použít 6 jedinečných kombinací, což je dostatečné množství pro omezený počet robotů použitých při této diplomové práci. Přidáním jedné další barvy již ale počet kombinací naroste na 12, což je více, než je používáno běžně ve *fotbalu LEGO robotů*. Pokud by bylo nutné, lze však také problém vyřešit zvýšením počtu použitých *LED diod*, jak bylo zmíněno výše.

Jedna z výhod tohoto typu panelu je, že byl vytvořen tak, aby *LED diody* bylo možné jednoduše vyjmout a nahradit *LED diodou* jiné barvy. Oprava nebo změna značení robota tudíž není nijak časově ani technicky náročná. Nevýhodou, kterou toto řešení přináší je, že *LED diody* jsou ne vždy panelu zasazeny dostatečně pevně a někdy tak při pohybu nebo

kolizi robota dochází ke ztrátě kontaktu a *LED dioda* přestane svítit, dokud ji uživatel opět správné nezastrčí do jejího slotu.

Další nevýhodou této metody bylo, že v obrazu kamery, která byla pro tento typ značení použita, světlo některých *LED diod* bylo příliš jasné a vytvářelo v obraze bílé body, které byly obklopeny prstencem barvy dané *LED diody*. Tento efekt zjednodušoval nalezení *LED diod*, dále pak ale velmi komplikoval identifikaci jejich barev, protože barevný prstenec byl výrazně ovlivněn jak barvou pozadí, tak světlem vyzařovaným z ostatních *LED diod*.

Tyto nevýhody identifikačního značení pomocí *LED diod* znamenaly, že došlo na zvažování úpravy použitého značení nebo dokonce nahrazení značením jiného typu.

3.2.2 LED diody s velkou plochou a nízkým profilem

Pokud by bylo možné použít *LED diody*, které by byly větších rozměrů, co se týče výšky a šířky (nikoli však hloubky) a ideálně čtvercového tvaru s plochou horní hranou, za předpokladu, že by svítivost těchto *LED diod* nebyla příliš vysoká a tudíž by výrazně neovlivňovala světlo vyzařované sousední *LED diodou*, mohli bychom umístit všechny tři *LED diody* těsně vedle sebe, tak aby se navzájem dotýkaly. Nehrozilo by tedy překrytí některé *LED diody* jinou, která je blíže k pozorovateli nebo kameře. Myšlenka je to pěkná, k jakému závěru jsem ale nakonec došel?

Tato možnost však byla zavržena z toho důvodu, že *LED diody* tohoto charakteru se buďto nevyrábí nebo nejsou k dostání běžnými způsoby. Další problém, který se projevuje u běžných *LED diod*, leží také v tom, že to jakou bude mít *LED dioda* barvu je určeno sklíčkem, které obaluje tu část *LED diody*, která emituje světlo. Znamená to tedy, že obal, který určuje tón a intenzitu barvy by u takové *LED diody* měl mnohem menší efekt na jas diody a na zachyceném snímku by se tedy projevil velkým bílým flekem, který by dále ztížil identifikaci barvy *LED diody*.

Druhým problémem je, že na obraze zachyceném námi použitou kamerou by světlo z jedné *LED diody* ovlivňovalo světlo z *LED diody* sousední a vznikla by tak barva, kterou by *analýza obrazu* nebyl schopna detekovat.

Od tohoto možného řešení se tedy nakonec z výše uvedených důvodů upustilo. Je ale možné, že by se v budoucnu dalo použít takové *LED diody* nebo kameru, které by umožnily řešení tohoto typu použít.

3.2.3 LCD panel

Další možností značení robota, kterou jsem zvažoval, bylo umístění *LCD* panelu na horní část robota. Pro použití takového panelu je však zapotřebí mít připojenou ovládací jednotku, která dokáže na panel vykreslit požadovaný obrazec. Tudíž by bylo potřeba takovou jednotku přidat nebo jí nahradit tu stávající. Nová ovládací jednotka by pak ale musela být kompatibilní s použitými periferiemi. Navíc snímání *LCD* panelu kamerou mívá obvykle negativní vliv na výsledný obraz. Za takového předpokladu ani toto řešení není vhodnou cestou ke zlepšení identifikačního značení robotů.

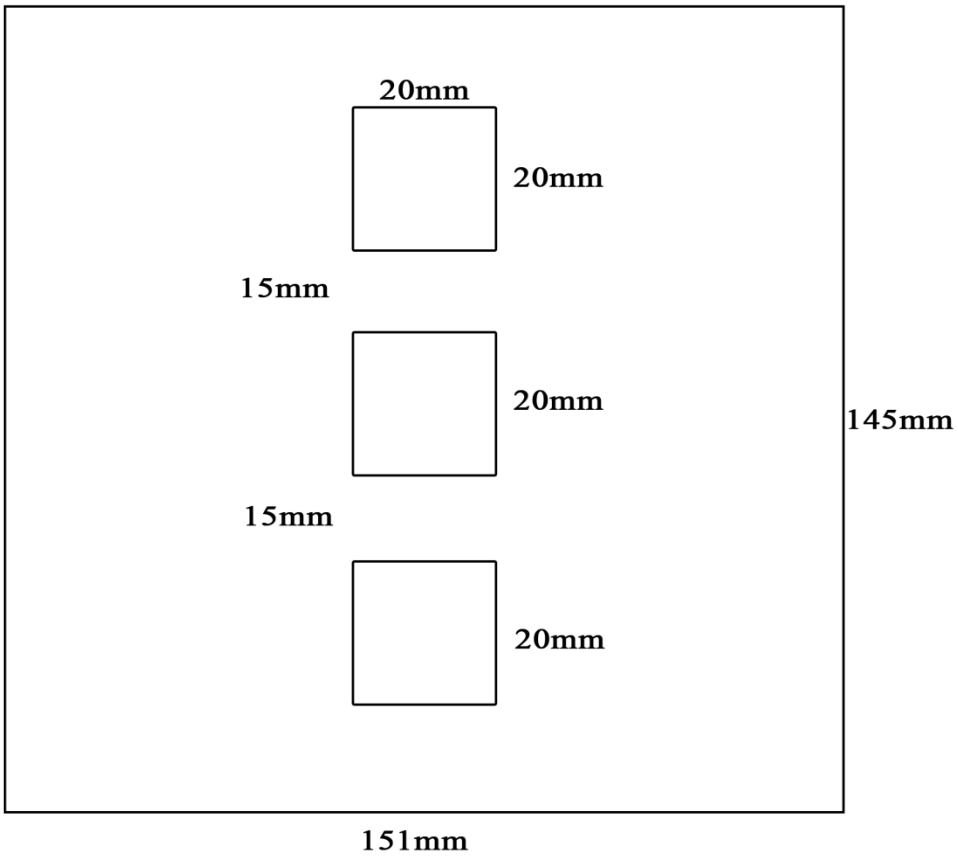
3.2.4 Barevné štítky

Použití aktivního světelného prvku jako například výše zmíněné *LED diody* může mít za následek, že světlo, které tento prvek emituje je ovlivněno světelnými podmínkami okolí, světlem jiného podobného prvku nebo barvou pozadí. Aby se předešlo nepříjemným efektům, bylo proto zvoleno jako další možný způsob identifikace, značení pomocí jednoduchých papírových štítků zvolené barvy.

Tento způsob značení je výhodný z toho hlediska, že barva značek není natolik ovlivňována okolním světlem. Navíc u značení pomocí *LED diod* nebyla použita modrá barva, protože *LED dioda* modré barvy vyzařovala příliš jasné světlo a na zachyceném obrazu pak již nebylo vůbec možno její barvu identifikovat. Tento problém však u modrých štítků nevzniká.

Naopak použití modré barvy namísto žluté se jeví jako velmi výhodné. V barevném formátu Hsv, který bude popsán dále, je totiž modrá barva snadno rozlišitelná od červené a modré, takže nedochází k nesprávnému určení barvy nalezeného identifikátoru.

U tohoto typu identifikace se někdy stávalo, že mezi identifikátory stejné barvy nebyl nalezen předěl a tak byly dva identifikační štítky považovány za jeden. Tento problém se podařilo vyřešit zvětšením mezery mezi jednotlivými identifikačními štítky. Stále však musela být zachována vzdálenost krajních štítků od okraje robota, aby nedošlo k tomu, že systém k sobě spojí nálezy, které k sobě nepatří. Následující obrázek ukazuje rozložení jednotlivých identifikačních štítků.



Obrázek 10: Rozložení a rozměry identifikačního značení na krytu robota, Zdroj vlastní

3.2.5 Vyhodnocení

Závěrem zhodnocení jednotlivých možností je, že způsob, který využívá barevných štítků, vykazuje nejméně vedlejších efektů a nejvyšší úspěšnost detekce robotů. Proto bylo dále používáno toto značení.

Nevýhodou tohoto značení je, že jej nelze použít při velmi nízkém osvětlení místnosti. Přesto je však primární využití tohoto projektu zaměřeno na situace, kdy osvětlení hrací plochy nedosahuje extrémně vysokých nebo nízkých hodnot. Navíc je tento typ značení nejblíže k původnímu značení použitému u *fotbalu robotů*.

3.2.6 Další možnosti značení

Existují ještě další možnosti značení. Například *IR LED diody* jsou také používány pro aplikace, které vykonávají analýzu obrazu. Otestování tohoto typu identifikace robotů by mohlo být námětem budoucích prací.

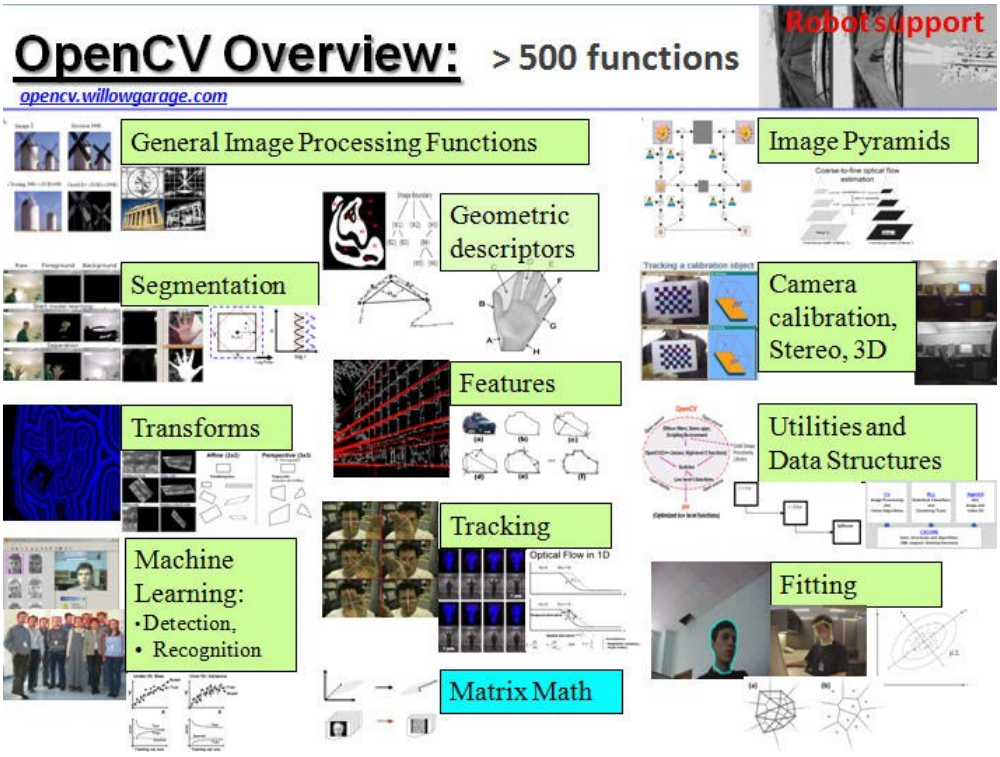
Během mého studia na Japonské univerzitě *Yokohama National University* jsem se setkal s možností rozpoznávání robotů pomocí bezdrátového signálu. Tato možnost nevyužívá kamery, a proto by tento způsob mohl znatelně snížit náročnost výpočetních operací, které musí vykonat počítač. Student, který již dříve podobnou technologii používal, mi potvrdil, že tento způsob je proveditelný, bylo by však potřeba upravit některé používané algoritmy výpočtu polohy, aby dokázaly také rozpoznat natočení robota. Metody jako TimeOfArrival (TOA), TimeDifferenceOfArrival (TDOA), and AngleOfArrival (AOA) by bylo možné pro realizaci tohoto značení použít. Z výše zmíněných je potom často používanou technologií s velkým množstvím dostupných informací metoda TDOA. Tuto metodu tedy také dále doporučuji k dalšímu výzkumu.

4 Kamerový systém

4.1 OpenCv

OpenCv znamená **Open** Source **C**omputer**V**ision. *OpenCv* jeknihovna programovacích funkcí pro zpracovávání obrazu v reálném čase. Knihovna obsahuje více než 2500 optimalizovaných algoritmů. Algoritmy mohou být použity k rozpoznávání tváří, identifikování objektů, určování činností lidí na videu, sledovat pohyb kamery, sledovat pohybující se objekty a mnoha dalším účelům v oblasti počítačového vidění[11].

OpenCv oficiální webové stránky se chlubí odhadem více než 5 miliónů stažených kopií. Společně s celou řadou jmen různých slavných společností, které jejich knihovny využívají. *OpenCv* obsahuje rozhraní pro *C++*, *C*, *Python* a *Javu* a podporuje *Windows*, *Linux*, *Android* a *Mac OS*.



Obrázek 11: Přehled základních funkcí *OpenCv*, Zdroj[12]

Knihovna *OpenCv* je základním stavebním kamenem aplikace sloužící k analýze obrazu, kterou jsem dostal od svého předchůdce. Vzhledem k tomu, že *OpenCv* obsahuje rozhraní pro *C++*, je výhodné psát program pro analýzu obrazu také v *C++*. Při vývoji aplikace v jazyku *C++* jsem se však dostal do problémů v oblasti propojení *C++* aplikace analýzy obrazu s *C#* aplikací strategií robotů. Více v sekci *C++/CLI*.

4.2 C++/CLI

Propojení dvou programovacích jazyků a to obzvláště u jazyků *C#* a *C++*, které fungují každý na jiném principu, se může zdát jako nesmysl. *C#* aplikace běží na virtuálním stroji zvaném *CLR*. Zdrojový kód je přeložen nejprve do kódu virtuálního stroje a pomocí metody nazývané *Just-in-time* kompilace jsou jednotlivé části kódu přeloženy do strojového jazyka počítače až ve chvíli, kdy je daný kód zapotřebí. Naproti tomu jazyk *C++* je vytvořen tak, že během kompilace je zdrojový kód přeložen rovnou do strojového jazyka počítače.

I přes to, že je mezi těmito konkrétními jazyky zásadní rozdíl byl vytvořen způsob jak je použít společně v jedné aplikaci a tak vytvořit projekt složený z obou typů kódu. Touto metodou je *C++/CLI*. *C++/CLI* je specifikace jazyka vytvořena Microsoftem. *C++/CLI* tak vytváří most, který propojuje rozdílné přístupy. *C++/CLI* vychází syntaxí z klasického *C++*, v některých ohledech se však liší.

Při použití této metody propojení *C++/CLI* a *C#* je myšlenkou vytvořit *v C++/CLI wrapper*, neboli obal okolo nativní *C++* aplikace, který bude obsahovat jakési rozhraní (nezaměňujte s programátorským pojmem interface), které je přístupné z vnější a C# aplikace tak může metody tohoto rozhraní volat jako by byly její vlastní. V metodách tohoto rozhraní *C++/CLI* může vytvářet instance a volat metody nativního *C++* kódu. Tím je náš spojovací řetěz dokončen.

V této fázi projektu se však i přes rozsáhlou snahu nepodařilo úspěšně *C++/CLI wrapper* implementovat. Protože použití *C++/CLI* nebylo podmínkou danou v zadání této práce, byl

zvolen odlišný přístup ke sjednocení celé aplikace. K analýze obrazu však bylo stále využito *OpenCv*, proto bylo potřeba nalézt způsob jako použít *OpenCv* knihovny v programovacím jazyku *C#*. K tomu posloužilo *Emgu CV*. O tom co je *Emgu CV* bude pojednávat následující kapitola.

4.3 Emgu CV

Emgu CV je způsob jako použít *OpenCv* pro programovací jazyky rodiny *.NET*. *Emgu CV* je totiž již zmíněný *wrapper* pro knihovny napsané v kódu jazyka jiného typu, konkrétně pro již popsanou knihovnu *OpenCv*.

Nabízí tak většinu funkcí jako originální knihovny *OpenCv* a uživatelé mohou vycházet z použití původních funkcí, přestože způsob volání těchto funkcí se v *Emgu CV* mírně liší. Na rozdíl od *OpenCv*, kde původní i výsledný obrázek musel být vložen do funkce jako parametr, v *Emgu CV* lze volat metodu pro úpravu obrázku na objektu představující obrázek a výsledný obrázek je zase návratovou hodnotou této metody. Pro srovnání poslouží ukázku kódu pro rozmazání obrázku:

```
// Řádek 1: Gausovské rozmazání v OpenCv (C++)
cv::GaussianBlur(src, dst, Size(3, 3 ), 0, 0 );

// Řádek 2: Gausovké rozmazání v Emgu CV (C#)
dst = src.SmoothGaussian(3);

// Řádek 3: Gausovské rozostření bez návratové hodnoty v Emgu CV (C#)
img._SmoothGaussian(3);
```

Výpis 1: Gaussovské rozostření za pomoci *OpenCv* v *C++* a *Emgu CV* v *C#*

Na prvním i druhém řádku kódu je *src* vstupní obrázek, *dst* je výsledný obrázek a následující parametry jsou koeficienty pro způsob a sílu rozmazání. Na prvním rádku je tedy vidět, že je nutné zadat do parametrů metody i výsledný obrázek. Na druhém řádku postačuje jeden parametr, a výstupní obrázek je výsledkem volané metody. Parametrů však lze vložit více po vzoru *OpenCv*. *Emgu CV* navíc ještě u řady metod umožňuje, aby metoda nevracela návratovou hodnotu, ale aby přímo upravovala obrázek, nad kterým je zavolána. Taková metoda je značena na začátku názvu podtržítkem a je demonstrována na řádku 3.

Emgu CV způsobilo výrazné zrychlení při práci na projektu díky použité syntaxi, ale navíc znamenalo čisté spojení mezi jednotlivými částmi projektu, které díky tomu ještě více působí a chovají se jako jeden celek namísto propojování samostatných částí.

4.4 Analýza obrazu

Analýza obrazu je, jak už byla představena v předchozích částech, softwarová část projektu, která použije zachycený snímek k tomu, aby identifikovala míč a hráče, následně zjistila informace jako jejich polohu vůči hřišti, směr robota a jeho hybnost a poté tyto informace předala dál *modulu strategií*.

Původní aplikaci vytvořenou Zdeňkem Neustupou, kterou napsal v jazyku *C++* za použití knihoven *OpenCv* a *Qt* jsem již představil. Aplikace byla původně navržena pro maximální rychlost zpracování obrazu a to takovým způsobem, aby dobře fungovala jako samostatný celek. Jejím záměrem však nebylo propojení s dalšími aplikacemi a tím spíše ne s aplikací napsané v jiném programovacím jazyku.

Obrázek 12: Snímek zachycený kamerou před provedením analýzy obrazu, Zdroj vlastní

Tato nevýhoda a nevýhody, které byly rozebrány v předchozích kapitolách, znamenaly, že tato aplikace nebyla použita a namísto toho byla vytvořena zcela nová aplikace v jazyku *C#* za pomocí *Emgu CV*, která však využívala některé z postupů, které byly v původní aplikaci použity. Vzhled a funkce aplikace budou rozebrány v kapitole 8. Následující část se věnuje samotnému procesu rozpoznávání objektů v obraze.

Před tím, než se na zachyceném obrázku provádí samotné operace pro vyhledávání objektů, je třeba provést pár základních operací. Které to jsou? To bude popsáno v kapitole „Základní funkce pro zpracování obrazu“. Před tím je však ještě třeba vysvětlit základní barevné formáty, které jsou v *Emgu CV* používány. Tomu se bude věnovat v kapitole „Použité barevné formáty“.

4.4.1 Požité barevné formáty

I přes to, že je možné v *Emgu CV* vytvořit obrázek pomocí metod, které patří *OpenCv*, *Emgu CV* také obsahuje definici vlastního objektu a doporučuje jej používat. Díky tomu používám pro vytvoření obrázku a určení jeho barevného formátu generickou třídu, která je definována jako *Image<TColor, TDepth>*,kde *TColor* určuje způsob jakým je uložena barva obrázku v paměti a *TDepth* určuje, jaký rozsah hodnot může jednotlivá buňka obsahovat. Každý z těchto parametrů může nabývat celé řady hodnot, které jsou vyčteny na oficiálních stránkách *Emgu CV*[13].

Ve výsledné aplikaci je pro parametr *TColor* využito pouze tři z existujících možností. První z nich je *Gray*. Jak je z názvu již jasně patrné, slouží tento objekt k reprezentaci pixelů obrázku v odstínech šedi. Data jsou v tomto případě uložena jako jedna matice hodnot a obrázek je tak reprezentován jen jedním kanálem. Konverzi obrázku mezi různými formáty provedeme jednoduše pomocí následujícího kódu:

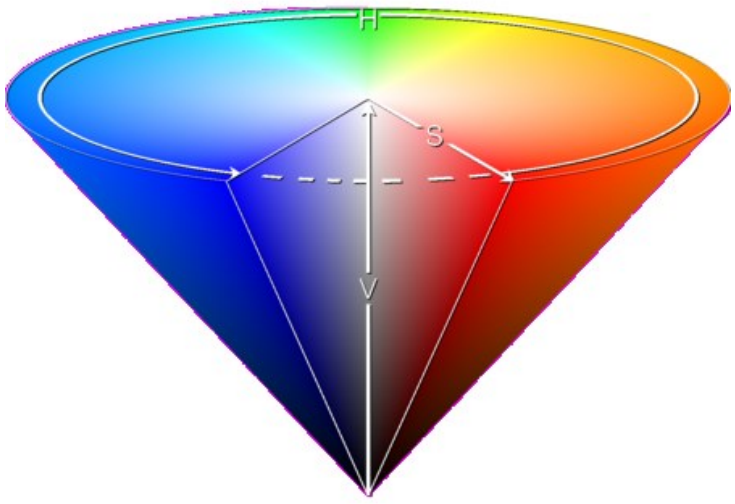
```
// Převod obrázku do odstímů šedí
Image<Gray, Byte>grayImg = currentFrame.Convert<Gray, Byte>();

// Převod obrázku do barevného format Bgr
Image<Bgr, Byte>bgrImg = currentFrame.Convert<Bgr, Byte>();

// Převod obrázku do barevného formátu Hsv
Image<Hsv, Byte>hsvImg = currentFrame.Convert<Hsv, Byte>();
```

Výpis 2: Převod obrázku do různých barevných formátů

Další možností jak určit typ barvy je použít *Bgr* formát. Běžně známý barevný formát je *Rgb*, kde každá buňka obrázku je reprezentována třemi kanály, kdy každé písmeno této zkratky představuje název daného kanálu. Tedy *R*: red - červená, *G*: green – zelená, *B*: blue – modrá. *Bgr* formát je s *Rgb* téměř identický pouze obsahuje jednotlivé kanály v jiném pořadí. Výhodou tohoto barevného formátu je, že lze relativně snadno identifikovat výraznou červenou, zelenou a modrou barvu. Nevýhodou však je, že pokud se barva míchá s jinou

nebo přímo je jiná než jedna ze tří výše zmíněných, je náročné ji z tohoto formátu identifikovat. Proto při vyhledávání barev nepoužíváme tento formát, ale využíváme formát s názvem *Hsv*[14].

Zkratka *Hsv* zastupuje slova *Hue*, *Saturation*, *Value*. *Hue* reprezentuje barevný tón. To znamená, že vlastně určuje barvu, díky čemuž můžeme vyvodit, že v daném rozsahu hodnot *Hue* se jedná o jistou barvu. Stejným způsobem lze určit celou řadu barev. Přesto však přechody mezi těmito barvami jsou plynulé, což znamená, že nemůžeme přesně určit, kde která barva začíná a kde končí, můžeme se tomu jen přiblížit, což může přinášet další nevýhody. Saturation představuje sytost, což znamená, že rozhoduje zda se vykreslí na obrázku sytě červená, nebo téměř šedá barva. *Value* nakonec určuje jas. Díky tomu je rozpoznávání jednotlivých barev obvykle mnohem jednodušší při použití *Hsv* než při použití *Bgr*, proto při identifikování barevných *LED diod* v zachyceném obraze nejprve dochází k převodu obrazu do tohoto formátu.



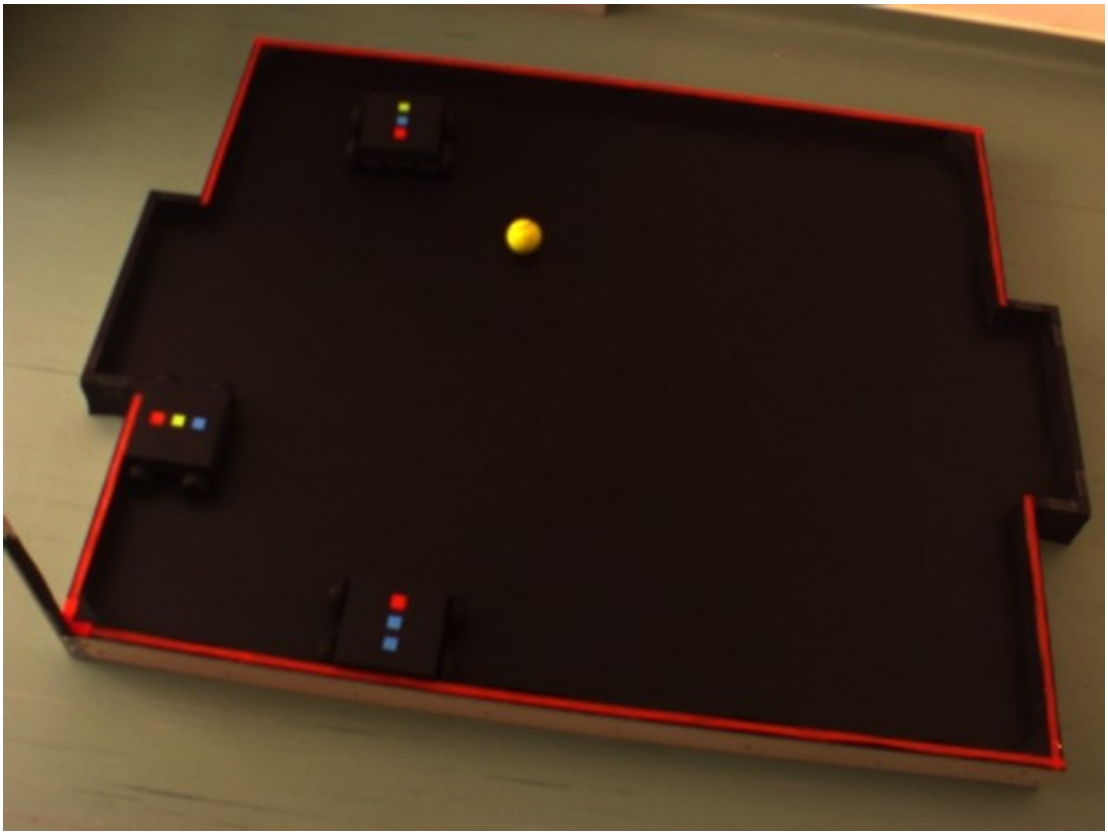
Obrázek 13: Grafické znázornění barevného formátu *Hsv*, Zdroj[15]

Jak je vidět z obrázku 11, použití kombinace barev modrá – červená - zelená je u tohoto modelu ideální případ, protože hodnota *Hue* každé barvy je nejvíce vzdálena ostatním dvěma a tudíž je pravděpodobnost zaměnění barvy za jinou z této kombinace minimální možná v rámci *Hsv* modelu.

4.4.2 Automatická a manuální kalibrace hřiště

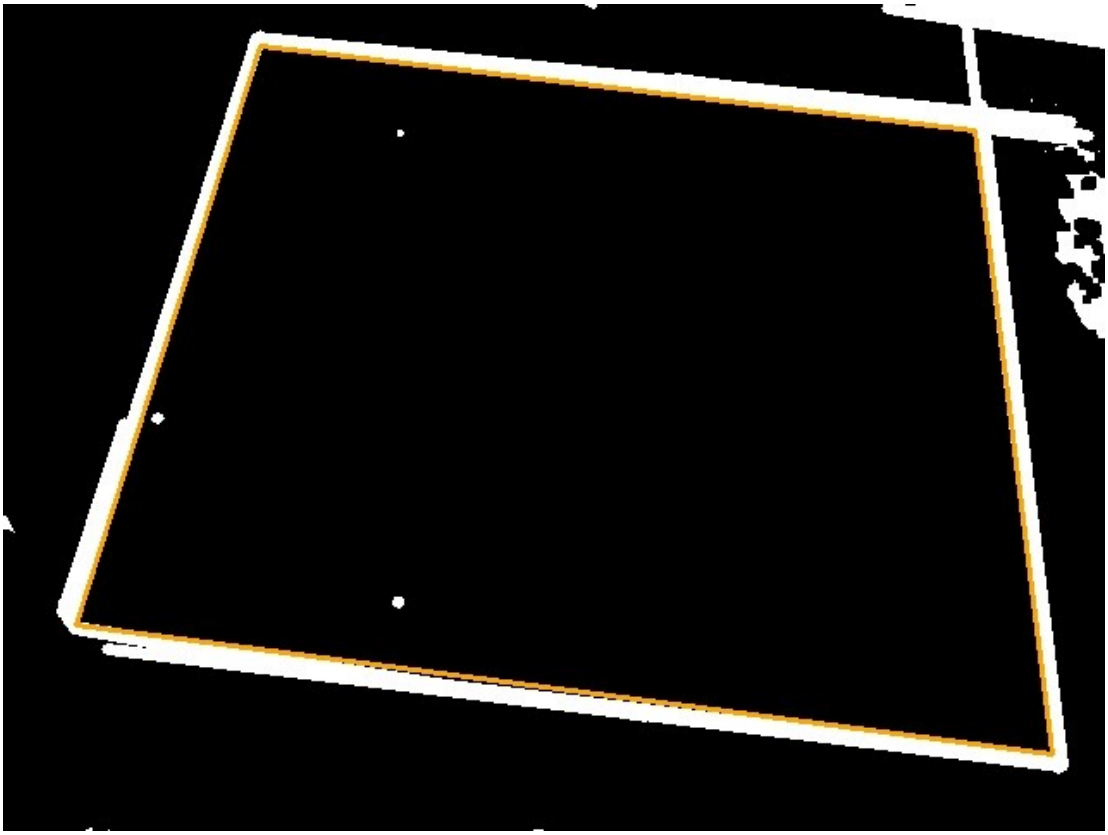
Ještě před tím, než aplikace začne provádět nad snímkem jakékoli operace, je třeba provést kalibraci hřiště. Kalibrace hřiště probíhá manuálním nebo automatickým způsobem. Manuální kalibrace je poměrně jednoduchá. Uživatel nejprve stiskne na klávesnici klávesu *Ctrl* a poté myší klikne postupně od levého dolního rohu v protisměru hodinových ručiček na všechny čtyři rohy hřiště. Při manuální kalibraci není třeba, aby v rozích hřiště byly umístěny jakékoli značky. Je to však pohodlnější při výběru jednotlivých rohů. Samozřejmou podmínkou pro správnou kalibraci hřiště a spuštění hry je, že na kameře je stále viditelné celé hřiště včetně rohů. Dále také, že pozice a natočení hřiště zůstává po celou dobu stejné. Pokud dojde ke změně těchto parametrů, je potřeba kalibraci zopakovat.

Druhý způsob, jakým je možné provést kalibraci hřiště, je automatický. Pro účely automatické kalibrace byla v pozdější fázi projektu umístěna podél hlavní části hrací plochy na mantinel hřiště umístěna oranžová páska. Na obrázku 14 je vyobrazeno, jak takto označené hřiště vypadá na snímku pořízeném kamerou.



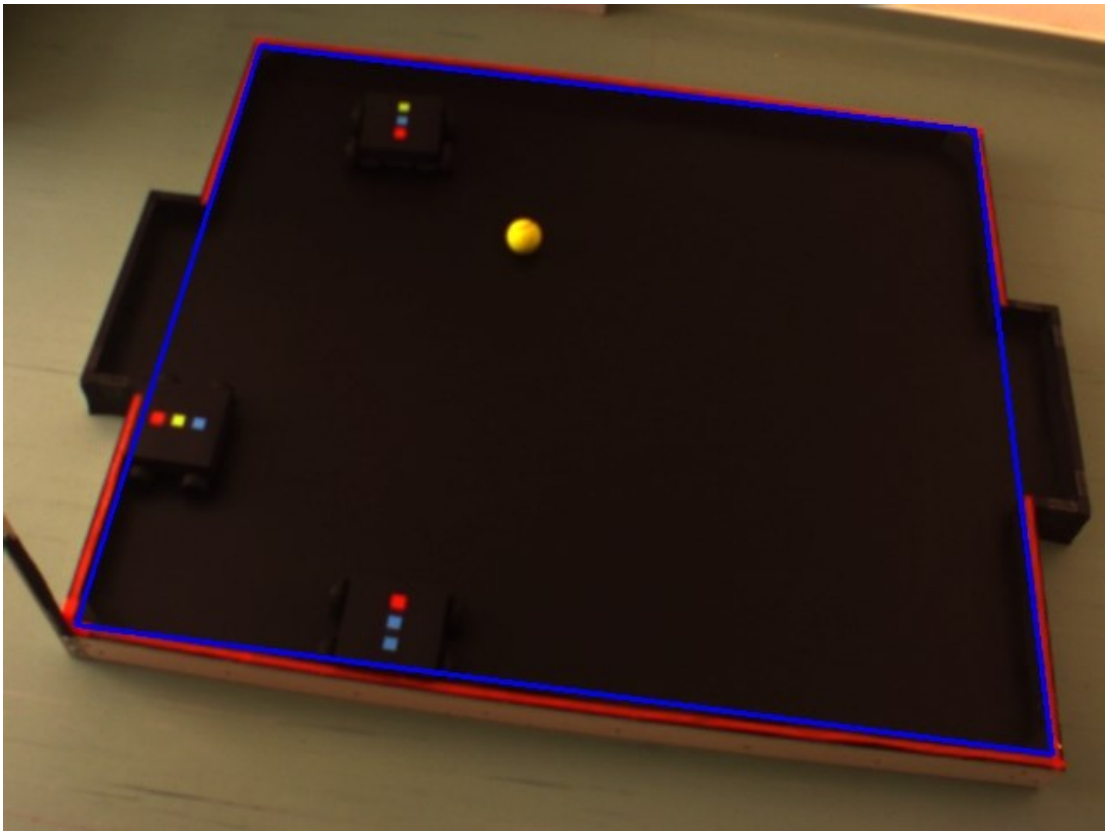
Obrázek 14: Značení hřiště použité pro automatickou kalibraci, Zdroj vlastní

Analýza obrazu zpracuje tento snímek během automatické kalibrace takovým způsobem, že vyhledá v obraze objekty oranžové až červené barvy a pixely všech ostatních barev začerní. Odstín barvy okraje ve výsledném snímku je náchylný na osvětlení a nastavení jasu kamery, obvykle proto není ve výsledném snímku vidět celý okraj hřiště, ale některá místa se ztrácejí. Zároveň ale dochází k zachování objektů, které nejsou součástí hřiště, ale spadají do přednastaveného barevného rozsahu. *Analýza obrazu* proto vyhledá ve snímku rovné čáry a protáhne je, aby byly výraznější, a aby došlo k úplnému uzavření vykresleného okraje hřiště. Dále následuje proces, který vyhledá v upraveném snímku kontury a pokusí se vyhledat čtyřúhelník. Výsledek těchto operací je vyobrazen na obrázku 15.



Obrázek 15: Zpracovaný snímek za účelem automatické kalibrace, Zdroj vlastní

Ve chvíli, kdy se *analýze obrazu* podařilo vyhledat čtyřúhelník, který je definován čarami, získá z těchto čar rohové body, u kterých identifikuje, zda se jedná o levý nebo pravý a horní nebo dolní roh čtyřúhelníku. Pomocí těchto bodů vykreslí nalezený čtyřúhelník do obrazu videa modrou barvou, jak je vidět na obrázku 16 a počká na odezvu uživatele. Pokud byla detekce hřiště neúspěšná, může uživatel proces opakovat nebo zvolit manuální kalibraci. Pokud byla kalibrace úspěšná, jsou body uloženy a následují operace popsané v následující kapitole.



Obrázek 16: Vyobrazení výsledku automatické kalibrace jak jej vidí uživatel, Zdroj vlastní

4.4.3 Perspektivní transformace a maska hřiště

Pokud došlo ke správné kalibraci hřiště, provádí aplikace následně dvě operace, které jsou zcela nové oproti aplikaci, kterou jsem přejal od svého předchůdce. První operace je nazvaná *perspective transformation* a druhou operací je vytvoření masky snímku.

Při pohledu hřiště člověk vidí, že základní plocha hřiště (bez vykrojení branek) je obdélníkového tvaru umístěný v prostoru, kdy protilehlé strany jsou rovnoběžné a rohy hřiště svírají úhel 90°, alespoň do té míry, kdy pouhým okem nerozeznáme, zda se skutečnost liší. Kamera ale postavená není postavená přímo nad středem hřiště a pod pravým úhlem vůči jeho ploše. Proto tvar, který hřiště nabývá na zachyceném snímku, není obdélník, ale konvexní čtyřúhelník. *Perspective transformation*, neboli perspektivní transformace, je transformace obrazu, kde za pomocí čtyř bodů počátečních a čtyř bodů cílových dochází ke

změně tvaru obrazu tím způsobem, že konvexní čtyřúhelník v obrázku poté opět vypadá jako obdélník, nebo jiný typ čtyřúhelníku jehož tvar a rozměry jsou zvoleny sérií cílových bodů. Provedení této transformace následně umožní získat souřadnice identifikovaných objektů z obrazu jednoduchým zpracováním x-ové a y-nové souřadnice objektu v obrázku.

Druhou zmíněnou operací je jednoduché maskování obrazu. Znamená to, že se nejprve vytvoří prázdný obrázek, o stejných rozměrech jako zachycený snímek, do kterého se vykreslí obdélník. Obdélník, který je použit v tomto projektu je má přednastavené parametry a velikostně i pozicí odpovídá tomu, na který byl již před tím přetransformován obraz hřiště. Pomocí této metody je tedy zajištěno, že při následujícím zpracovávání snímku nebylo třeba provádět filtry obrazu na částech snímku mimo samotné hřiště. Na obrázku 17 je ukázáno, jak vypadá snímek po provedení perspektivní transformace a maskování. Dále jsou na něm zvýrazněny hrany hřiště. Operace byly provedeny nad obrázkem 12.

Obrázek 17: Výsledek provedení maskování a perspektivní transformace, Zdroj vlastní

4.4.4 Základní funkce pro zpracování obrazu

Při zachycení obrazu kamery jsou jednotlivé pixely samostatnými prvky. Není to tak, že když je zachycen objekt, tak pixely znázorňující tento objekt by měly všechny stejnou barvu, na sebe navazovaly nebo měly nějaký ukazatel toho, že k sobě patří v rámci toho objektu. Naopak je to mix různých odstínů a sytostí dané barvy a někdy dokonce i barvy jiné. Lidské oko však takový obrázek vidí jako by všechny pixely daného objektu měly téměř stejnou barvu, případně že se odstín uvnitř objektu plynule mění.

Když bychom chtěli v takovém obrázku vyhledat objekt pomocí barvy, mohlo by se stát, že řada pixelů uvnitř objektu hledané barvy by nebyla nalezena, protože ve skutečnosti nabývají barvy jiného odstínu, či je to šum, který je na snímcích běžně přítomen. Zároveň by proces nalezl řadu jiných pixelů, které jsou pouze šumové i přes to, že objekt, ve kterém byly nalezeny, by měl ve skutečnosti jinou barvu.

Toto však není neřešitelný problém. Právě naopak, lze jej alespoň částečně vyřešit relativně jednoduchým způsobem. K tomu účelu slouží jemné vyhlazení, neboli rozostření obrázku. Během tohoto procesu se vezme postupně hodnota každého pixelu obrázku a nahradí se hodnotou vypočítanou z pixelů, které jej obklopují. Tím se lze vyhnout extrémně rozdílným hodnotám sousedních pixelů a pixely uvnitř jednotlivých objektů se do jisté míry sjednotí. Při jemném rozostření nedojde k narušení hran nebo skokových přechodů mezi různými barvami, a vyhledávání hran a dalších objektů v obrázku je nakonec ještě snadnější než dřív.

Za účelem rozostření obrázku obsahuje *OpenCv* celou řadu funkcí, reprezentující různé matematické operace. Pravděpodobně nejužitečnější metoda pro vyhlazení obrázku, kterou obsahuje *OpenCv* je Gaussovské rozostření, které již bylo prezentováno v předchozí kapitole. Po provedení rozostření je zpracovávaný obrázek připraven k dalšímu kroku. To však samo o sobě ještě nestačí, aby bylo možné na obrázku efektivně a rychle vyhledávat objekty. Je potřeba převést jej do odstínu šedi.

Jak bylo uvedeno v předchozí kapitole, obrázek v odstínech šedi ukládá data v jednom kanálu oproti třem kanálům ve formátu *Bgr*. Vzhledem k tomu, že matematické operacemi nad

obrázkem je třeba provádět nad všemi kanály, pokud vykonáme tuto operaci nad obrázkem v odstínu šedi, pak ji pro celý obrázek vykonáme jen jednou, což je důležité vzhledem k výpočetní náročnosti některých úloh.

S obrázkem, který je v odstínech šedi a jemně vyhlazen už můžeme provádět další operace pro vyhledávání námi požadovaných objektů.

4.4.5 Vyhledání míče

Vyhledání kulatého míče v obraze není z programátorského hlediska záležitost dlouhého kódu. Po vyhlazení obrázku a převedení jej do odstínu šedé, je aplikována metoda *Canny*. Tato metoda vyhledá a vykreslí v obrázku hrany podle zadaných parametrů. Výsledkem je v tomto případě černobílý obrázek, kde černá barva znamená pozadí, a bílou barvou jsou vykresleny nalezené hrany. Tato funkce není v kódu zahrnuta, protože je již součástí metody, která v obrázku vyhledává kruhy.

Následně se v obrázku s vykreslenými hranami pokusíme najít kulatý míč. *Emgu CV* má pro tento účel vytvořenu funkci *HoughCircles*. To zda funkce dokáže v obrázku nalézt kruh, záleží na zvolených parametrech, ale vhodné nastavení nemusí být snadné najít. Parametry použité ve zvolené aplikaci parametry jen mírně lišily od parametrů, které ve své aplikaci použil Zdeněk Neustupa, který k vyhledávání míče přistupoval stejným způsobem.

```
// Převod do odstínů šedi a rozmazání
Image<Gray, Byte> ballSearchFrame = currentFrame.Convert<Gray, Byte>().SmoothGaussian(5);
// Vyhledávání kruhů pomocí metody HoughCircles
CircleF[] circles = ballSearchFrame.HoughCircles(
                        new Gray(220), // První hodnota pro thresholding
                        new Gray(30),  // Druhá hodnota pro thresholding
                        2.2, // Rozlišení akumulátoru pro určení středu kruhu
                        ballSearchFrame.Width, // minimální vzdálenost mezi dvěma nálezy
                        10, //minimální poloměr míče
                        25 //maximální poloměr míče
                        )[0]; // Uložení výsledků pouze z nultého kanálu

// Bere v úvahu výsledek pouze pokud byl nalezen právě jeden míč
if (circles.Length == 1)
{
    ball = new Core.Ball();
    ball.Position.X = circles[0].Center.X;
    ball.Position.Y = circles[0].Center.Y;

    // Kontrola zda se míč nachází v levé nebo pravé bráně
    if (field.IsPointInsideLeftGoal(ball.Position.ToPoint()))
    {
        IsBallInLeftGoal = true;
    }
    if (field.IsPointInsideRightGoal(ball.Position.ToPoint()))
    {
        IsBallInRightGoal = true;
    }
}
```

Výpis 3: Kód detekce míče

Úspěšnost detekce míče byla mírně snížena jinak velmi užitečnou funkcí perspektivní transformace, protože tato funkce nezachovává tvary a původně kulatý míč pak má na obraze tvar eliptický.

4.4.6 Vyhledání a identifikace robotů

Při spuštění aplikace předchůdce se ukázalo, že úspěšnost vyhledávání robotů touto aplikací byla velmi nízká. Po umístění tří robotů na hrací plochu aplikace dokázala identifikovat střídavě dvě až čtyři *LED diody* v jednotlivých snímcích, přičemž se většinou nepodařilo nalézt všechny *LED diody* alespoň jednoho robota. V daných světelných podmínkách, které

ukázaly být pro vyhledávání robotů na snímcích klíčové, se tedy ve většině případů nepodařilo identifikovat ani jediného robota na hrací ploše.

Kvůli této velmi nízké úspěšnosti vyhledávání a identifikace robotů výsledná aplikace této diplomové práce používá z větší části jiný algoritmus vyhledávání robotů. Vyhledávání robotů je rozděleno do tří oddílů: 1) vyhledání a zpracování kontur, 2) zpracování nálezů, 3) identifikace robotů. Následuje rozbor jednotlivých částí.

Vyhledání kontur je poměrně jednoduchá záležitost. Po již popsaném rozmazání obrazu dojde k převodu barev do formátu *Hsv*. Následně dochází ke klíčovým operacím. Nejprve je na všech pixelech obrázku provedena kontrola, zda odpovídají předem nastavenému rozsahu. Tato operace se provádí pro každou barvu zvlášť. Rozsah červené barvy je definován v tomto případě jako 0 – 15 a 160 – 180, neboli pro definování červené barvy byly použity dva rozsahy. Je tedy třeba provést operaci pro každý rozsah zvlášť a výsledný obraz spojit do jednoho pomocí jednoduché operace *Add*, která pouze přičte hodnoty pixelů jednoho obrázku k hodnotám pixelů druhého obrázku. Následně se na výsledném obrázku každé barvy provádí operace *FindContours,* která na každém obrázku vyhledá kontury a uloží je do kolekce kontur. Jednotlivé kolekce jsou pak předány metodě, která je zpracuje dle zadané barvy.

```
// Zachycený snímek je rozmazán a převeden do barevného format Hsv
Image<Hsv, Byte> img = currentFrame.SmoothGaussian(7).Convert<Hsv, Byte>();

// Rozsahy jednotlivých barev v Hsv formátu
Hsv greenLower = new Hsv(25, 20, 60);
Hsv greenUpper = new Hsv(65, 250, 250);

// Rozsah červené barvy je 0 – 15 a 160 – 180, každý rozsah je třeba defonovat zvlášť
Hsv red1Lower = new Hsv(0, 90, 90);
Hsv red1Upper = new Hsv(15, 255, 255);
Hsv red2Lower = new Hsv(160, 90, 90);
Hsv red2Upper = new Hsv(180, 255, 255);

Hsv blueLower = new Hsv(80, 10, 80);
Hsv blueUpper = new Hsv(110, 255, 255);

// Maskování jednotlivých barev a vyhledání kontur
Image<Gray, Byte> red = img.InRange(red1Lower, red1Upper);
red.Add(img.InRange(red2Lower, red2Upper));
Contour<Point> contoursRed =  red.FindContours(
                    Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_NONE,
                    Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_EXTERNAL);

Image<Gray, Byte> green = img.InRange(greenLower, greenUpper);
Contour<Point> contoursGreen = green.FindContours(
                    Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_NONE,
                    Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_EXTERNAL);

Image<Gray, Byte> blue = img.InRange(blueLower, blueUpper);
Contour<Point> contoursBlue = blue.FindContours(
                    Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_NONE,
                    Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_EXTERNAL);

// Průchod konturami každé barvy
blobs = new List<Blob>();
this.processContours(contoursRed, LedColor.RED);
this.processContours(contoursGreen, LedColor.GREEN);
this.processContours(contoursBlue, LedColor.BLUE);
```

Výpis 4: Kód detekce barevných štítků

Metoda zpracovávající kontury je všechny postupně projde a vyřadí ty kontury, které nemají správnou velikost a neleží uvnitř hřiště. Dochází ale k situacím, kdy nalezená kontura ve skutečnosti patří míči, proto navíc tato metoda vyloučí všechny nálezy, které jsou příliš blízko středu míče. Všechny kontury, které úspěšně prošly tímto filtrováním, jsou pak podle své barvy zařazeny do seznamu *blobů*, neboli nalezených identifikátorů.

```
using (MemStorage storage = new MemStorage())
{
    // Projití všech kontur
    for (; pContours != null; pContours = pContours.HNext)
    {
        // Zpracování aktuální kontury
        Contour<Point> currentContour = pContours.ApproxPoly(pContours.Perimeter * 0.05, storage);

        // Kontrola velikosti kontury
        if (currentContour.Area > 12 && currentContour.Area < 110)
        {
            // Kontrola zda kontura leží uvnitř hřiště
            if (field.IsPointInsideField(currentContour.BoundingRectangle.Location))
            {
                double dist = 30;
                if (ball != null)
                {
                        dist = Math.Sqrt( (currentContour.BoundingRectangle.Location.X - ball.Position.X)
                        * (currentContour.BoundingRectangle.Location.X - ball.Position.X) +
                        (currentContour.BoundingRectangle.Location.Y - ball.Position.Y) *
                        (currentContour.BoundingRectangle.Location.Y - ball.Position.Y));
                }
                // Kontrola zda kontura leží dostatečně daleko od středu míče
                 if (dist > 20)
                 {
                    Blob b = new Blob();
                    // Pozice nálezu je zadána podle středu kontury
                    b.Location = new Point(currentContour.BoundingRectangle.X +
                                                (currentContour.BoundingRectangle.Width / 2),
                                           currentContour.BoundingRectangle.Y +
                                                (currentContour.BoundingRectangle.Height / 2));
                    // Přiřazení barvy, která byla zadána v paramtru metody
                    b.Color = pLedColor;
                    blobs.Add(b);
                }
            }
        }
    }
}
```

Výpis 5: Kód filtrování nalezených kontur

Následně dochází k organizaci nálezů do skupin po třech podle vzdálenosti. Nejbližší nálezy totiž logicky patří stejnému hráči. Navíc je vzdálenost nálezů omezena maximální hodnotou, takže by se nemělo stávat, že robot, u kterého se nepodařilo identifikovat jeden nebo více nálezů, si přisvojí nalezený identifikátor jiného robota.

Následně je přiřazování nálezů robotům dokončeno jejich seřazením v rámci robota. Vzhledem k omezenému počtu robotů používá systém aktuálně u každého robota pouze jeden červený identifikátor, který slouží jako ukazatel směru robota. Nalezený identifikátor s červenou barvou je tedy umístěn na první místo v pořadí trojice a následně je jeho pozice porovnána se zbylými dvěma. Bližší identifikátor je zařazen jako *Blob2*, vzdálenější jako *Blob3*. Rozšíření této metody pro práce se dvěma červenými identifikátory není nijak náročným problémem pro možné budoucí rozšíření implementace.

Algoritmus identifikace jednotlivých robotů, který je prováděn následně je již triviálním porovnáváním barev identifikátorů nalezených robotů s barvami identifikátorů robotů, kteří byli nastaveni jako očekáváni hráči na hřišti. Při nalezení shody se pak robotovi nalezenému analýzou obrazu zapíše *Id* přednastaveného robota a především číslo portu, který se používá pro komunikaci prostřednictvím *Bluetooth*.

Tímto dochází ke spárování robota detekovaného v obraze s konkrétním, jemu odpovídajícím skutečným robotem na hřišti. Při nalezení jednotlivých objektů na hřišti je pak už jedinou úlohou této části aplikace uložení nalezených robotů do proměnné, aby v příštím cyklu bylo možné vypočítat pohyb hráče. Objekty robotů jsou poté předány *modulu strategií*, aby určilo kam se má který robot posunout.

Původní řešení počítalo s použitím *LED diod*. Při tomto řešení však bylo problematickým vlivem na úspěšnost nalezení robotů měnicí se, přirozené osvětlení. Za předpokladu, že hra neprobíhá v místnosti bez oken, kde je vždy stejná intenzita a barevný tón osvětlení, dochází k tomu, že přírodní zdroj světla může mít na obraz kamery každý den jiný vliv a dokonce se může změnit během okamžiku, když slunce zastíní mrak a místnost je najednou výrazně

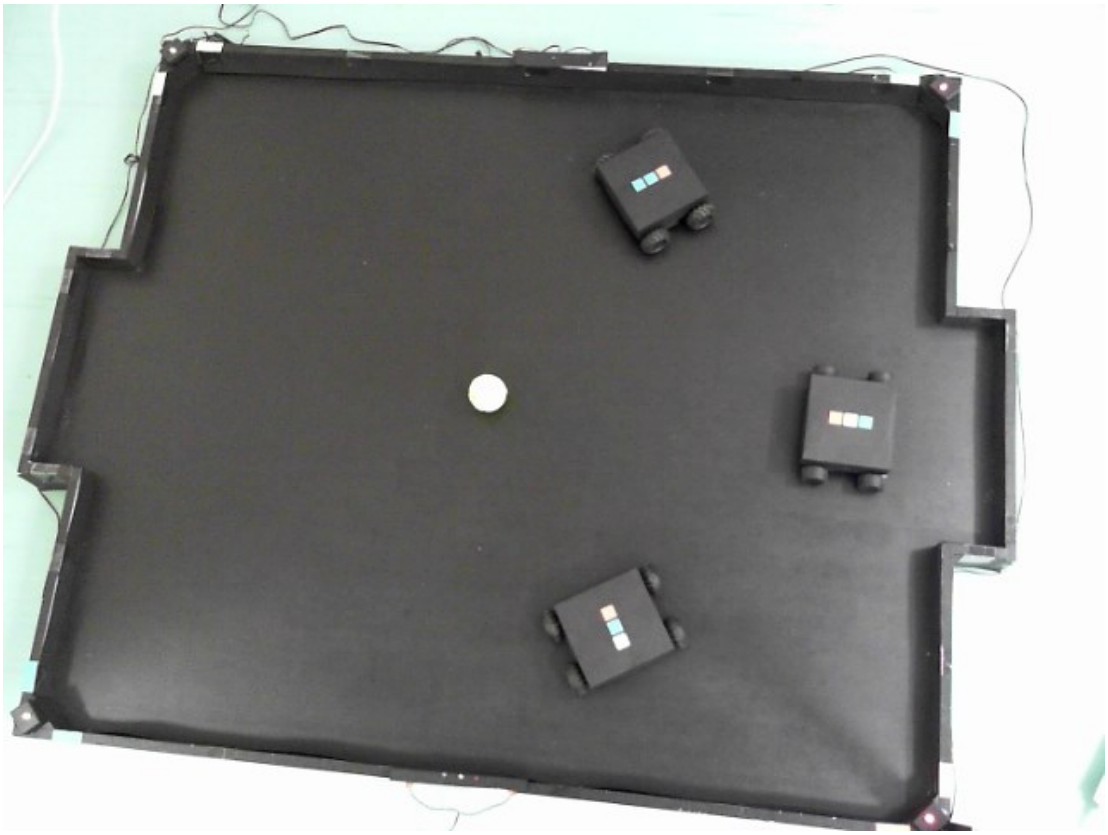
tmavší a světlo má jinou teplotu. Tón barev zachycených *LED diod* se tak může velmi snadno změnit tím způsobem, že již neodpovídá přednastaveným rozsahům barev.

Tento problém byl u barevných štítků z velké části odstraněn. Následná změna kamery dále zvýšila úspěšnost správné detekce robotů natolik, že při správném nastavení jasu a zaostření na objektivu kamery již nebylo zaznamenáno chybné vyhodnocení identifikace robota.

4.4.7 Použité kamery

Pro snímání byly použity postupně dvě kamery. První byla použita u řešení, které pro identifikaci využívalo *LED diod*. Takto kamera byla umístěná přibližně nad delší hranou hřiště. Tato kamera mimo automatické ostření a nastavení jasu neumožňuje žádné způsoby kalibrace nebo pokročilého nastavení. Navíc byla velmi citlivá na změnu osvětlení. V prostorách laboratoře, se osvětlení místnosti zvenčí velmi často měnilo co do jasu a teploty světla, i když byla snaha vliv vnějšího osvětlení minimalizovat zatažením žaluzií nebo zakrytím oken. Vliv změny světelných podmínek se pokaždé silně projevil na obrazu kamery, a tudíž došlo k negativnímu ovlivnění analýzy obrazu. Jemným vyladění parametrů v aplikaci analýzy obrazu obvykle úspěšnost identifikace robotů zvýšilo, ale bylo třeba vždy provést manuální úpravu přesně podle situace, která právě nastala. V některých případech byl vliv na obraz natolik velký, že barvu *LED diody* v obraze bylo téměř nemožné rozpoznat ani lidským zrakem.

Objevit *LED diody* v obraze není náročný proces a zde většinou problém nenastává. Tento problém s barevným podáním obrazu však výrazně komplikuje schopnost systému určit, jakou barvu konkrétní *LED dioda* vyzařuje. Kamera dále pravidelně vykazovala ten nedostatek, že se spouštělo automatické ostření a tak se stal obraz na nějakou dobu nepoužitelným, poté se opět vrátila do původního nastavení ostření.



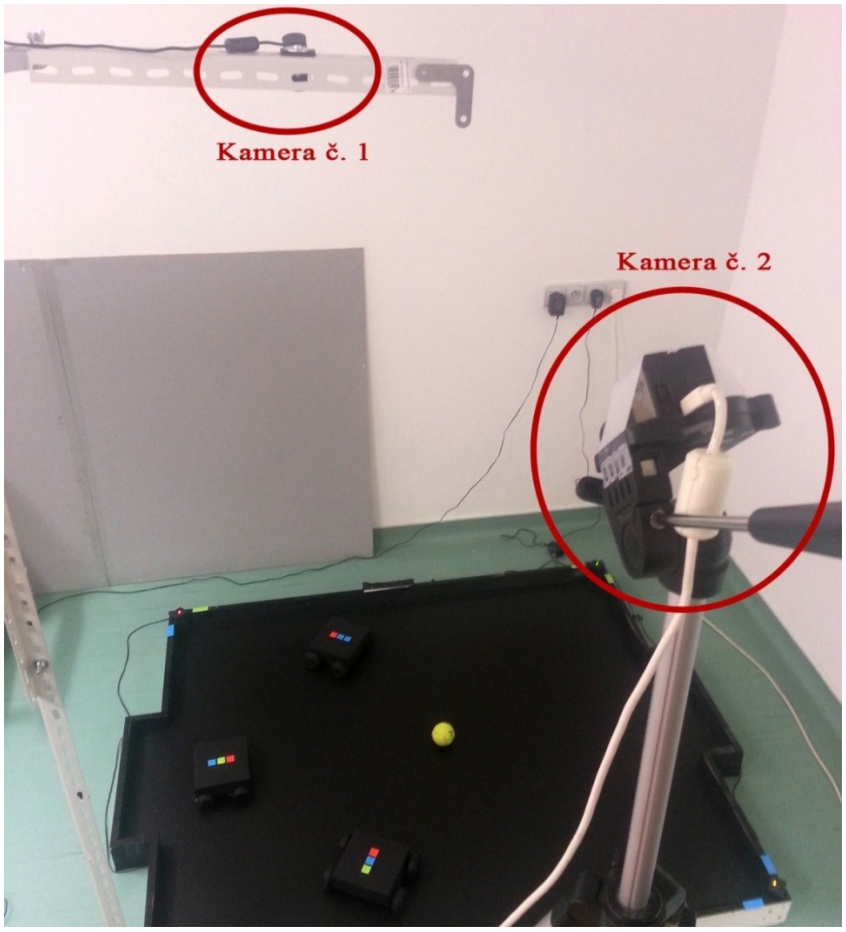
Obrázek 18: Barevné podání první kamery bylo nedostačující, Zdroj vlastní

Při použití této kamery při identifikaci robotů pomocí barevných štítků pak kamera nedokázala správně zachytit barvy. Barva použitých štítků tak pomocí této kamery byla výslednou aplikací neidentifikovatelná. Z uvedených důvodů se upustilo od použití této kamery.

Na řadu proto přišlo otestování druhé kamery, která je v laboratoři pro *fotbal LEGO robotů* dostupná. Tato kamera byla původně za účelem hry *fotbalu robotů* zakoupena. Tato kamera má však nastavenou pevnou ohniskovou vzdálenost a proto se ji nepodařilo připevnit na rám určený pro kameru tak, aby bylo v zachyceném snímku celé hřiště. K tomu do jisté míry přispívalo neobvyklé rozlišení této kamery. *OpenCv* bohužel podporuje jen málo různých rozlišení kamer. Při snaze zachytit obraz v požadovaném rozlišení, dochází ale k ořezu obrazu, což problém ohniskové vzdálenosti zhoršuje. Tento problém se nepodařilo vyřešit ani

úpravou nastavení pomocí originálních ovladačů pro tuto kameru, stále totiž neprobíhalo škálování obrazu na menší rozměr, ale pouze ořez.

Tento problém byl vyřešen použitím stativu, který byl postaven vedle hřiště, tak že hřiště bylo v zachyceném obraze vidět celé. Kameru bylo nutné velmi jemně vyladit, aby nedocházelo k chybné interpretaci barevných tónů během zachycení snímku. Ve chvíli kdy se světelné podmínky v místnosti změnily, stačilo na kameře jemně přenastavit světlost objektivu, tak aby systém dokázal identifikovat všechny roboty na hřišti a již bylo vše v pořádku a nebylo potřeba měnit nastavení parametrů aplikace. Výsledný obraz této kamery je dále zachycen na snímcích použitých při experimentech.



Obrázek 19: Umístění kamer vůči hřišti, Zdroj vlastní

Obrázek 19 ukazuje, jak byly kamery umístěny. Tento snímek byl zachycen na kameru mobilního telefonu Samsung Galaxy S3 a je vidět, že ostrost snímku i podání barev je na vysoké úrovni. Navíc telefony jako i tento disponují v dnešní době vysokým výpočetním výkonem. Proto by mohlo být zajímavé věnovat se v některé z budoucích prací vytvoření aplikace pro analýzu obrazu pro některý z mobilních operačních systémů.

5 Modul strategií

Strategické knihovny jsou nezbytnou součástí hry *fotbalu LEGO robotů*. Tento *modul strategií* jsem obdržel od vedoucího mé diplomové práce jakou součást *simulátoru hry LEGO robotů*. Z tohoto simulátoru byly vypreparovány kódy pro rozhodování, kam se mají roboti přesouvat a napojeny na aplikaci analýzy obrazu. Pro připojení bylo potřeba strategie správně nastavit a převést objekty, které byly získány analýzou obrazu na objekty, které používá tento *strategický modul*. Aby došlo k přenesení všech nezbytných informací z analýzy obrazu do strategických knihoven a následně do *Bluetooth* rozhraní, byly některé objekty *modulu strategií* rozšířeny.

Dále bylo potřeba vytvořit soubor s definovanými strategiemi, který je využíván *modulem strategií* při výběru výsledných pozic pro určení, kam se mají jednotliví roboti přesunout. Pro testovací účely byl vytvořen soubor strategií. Vzhledem k omezenému počtu skutečných robotů ale hra mohla být otestována maximálně se třemi hráči. Nejprve byla vyvinuta snaha hrát hru jeden proti jednomu, následně byl ale zaujat jiný postoj k nedostatku robotů. Vytvořený soubor proto obsahoval pravidla pro hru tři na tři. U tohoto řešení bylo počítáno s tím, že tři roboti, kteří byli k dispozici, hráli společně v jednom týmu, zatímco druhý tým byl ve skutečnosti prázdný, *modulu strategií* byly ale předány falešné souřadnice. Samotná struktura souboru strategií a způsob jakým jej používá *modul strateggií*, budou popsány v následující kapitole.

Modul strategií však nebyl plně připraven na použití v tomto projektu. Při spuštění aplikace proto došlo k chybě a *modul strategií* nefungoval podle očekávání. Problém tkvěl ve dvou zásadních bodech.

Problematickým bodem číslo jedna bylo to, že strategie automaticky vybrala z každého týmu jednoho robota, který sloužil jako brankář. Jedinou jeho funkcí je, že se pohybuje na okraji brány podle polohy míče tak, aby stál vždy mezi bránou a míčem. Při hře jeden ne jednoho však dva brankáři, kteří se vůbec nepohybují, nestačí k tomu, aby bylo možné provozovat hru *fotbal LEGO robotů*. Tento problém však měl jednoduché řešení, na konec seznamu robotů,

které jsou vkládány do *modulu strategií*, byl přidán jeden falešný robot, který je strategiemi považován za brankaře, ale není to hráč, který je skutečně postaven na hrací ploše. Skutečný hráč tak může díky tomu hrát roli, jakou mu strategie zvolí.

Po vyřešení prvního problému se ale objevil další, mnohem zásadnější problém. I přes to, že je v nastavení strategie použit parametr určující počet hráčů, je celá strategická knihovna vytvořena tak, že počítá s tím, že hra bude hrána pěti roboty v každém týmu. Pokud je robotů méně, proces načítání knihoven strategií selže. Protože je výjimka zachycena, ale není propagována dál, program dále běží, jakoby bylo vše v pořádku. V tomto případě bylo nutné chyby krok po kroku vystopovat.

Po opravě chyby načítání strategií přesto docházelo k vyvolání jiných výjimek, případně pádu programu. Postupně bylo tedy nutné upravit *modul strategií* tak, aby přijímal dynamický počet hráčů podle toho, kolik hráčů do něj bylo skutečně vloženo. Tedy nejen v procesu načítání textových strategií, ale také při jejich vyhodnocování, tj. v celém rozsahu, který byl v mém případu vyžadován.

Následně objevené chyby návrhu, jako například duplikace kódu, již nijak neomezovaly vykonávání hry a tak k žádným dalším změnám v *modulu strategií* již přistoupeno nebylo.

5.1 Struktura modulu strategií

Modul strategií je rozdělen do tří základních částí. První část je nazvána *Core* a obsahuje základní objekty, se kterými zbytek modulu pracuje. Zde jsou umístěny objekty například pro nastavení hry, objekty reprezentující roboty nebo míč. Dále je tady třída *Vector2D*, která je využívána pro reprezentaci pozice a pohybu robotů a míče. Jsou zde také další pomocné třídy a objekty umožňující, aby *modul strategií* běžel v samostatném vláknu. Dále je zde také důležitá třída *Storage.*

Storage slouží jako vstupní bod *modulu strategií* a obsahuje všechny údaje, které jsou během hry tomuto modulu předávány. Po vložení všech potřebných informací dochází k předání těchto hodnot do *modulu strategií*. Strategická knihovna přepočítá pozice všech objektů na

hřišti na gridové souřadnice a následně podle pravidel, která byla vložena prostřednictvím souboru strategických pravidel, vyhodnotí, jaké pravidlo nejvíce odpovídá dané situaci. Rozměry mřížky jsou definovány v souboru pravidel. Během vyhodnocování dochází k porovnávání vzdáleností jednotlivých robotů s pozicemi určenými jednotlivými pravidly. Pomocí tohoto mapování tedy strategická knihovna vyhodnotí, na které gridové souřadnice se má který robot přesunout.

Tyto získané gridové souřadnice jsou dále předány části, která je nazvána *taktická knihovna*. Úkolem této *taktické knihovny* je přepočítat gridové souřadnice zpět na skutečné souřadnice na hrací ploše. Tyto získané souřadnice jsou následně uloženy zpět do *Storage*, kde k nim může přistoupit již jiná část projektu, v našem případě rozhraní pro ovládání robotů pomocí *Bluetooth*.

5.2 Soubor pravidel

Soubor s definovanými pravidly strategie vytváří z *modulu strategií* dynamický program, který ve stejné situaci vyvodí jiné závěry podle toho, jaké údaje mu uživatel vloží bez toho, aby byl nutný jakýkoli zásah do kódu programu.

Soubor pravidel strategie není velmi komplikovaný. Na začátku souboru jsou uvedeny základní údaje o strategii. Jedním důležitým parametrem, který je zde zadán, je počet sloupců a řádků již zmíněné mřížky, která bude hřiště během hry rozdělovat. Vyšší hustotou buněk mřížky lze dosáhnout větší přesnosti robotů, na druhou stranu však i pravidla musí být přesnější a jejich počet narůstá.

Po úvodních informacích následuje seznam pravidel. Pravidla jsou složena z pěti řádků. První řádek obsahuje číslo daného pravidla, dále „o“, je-li to pravidlo útočné a nakonec název pravidla. Další řádky obsahují pozice vlastních robotů, pozice robotů protihráče a pozici míče. Poslední řádek obsahuje informaci, na které pozice se mají roboti přesunout. Za zmínku stojí, že pravidla obsahují o jednu pozici méně než je použitý počet robotů. Poslední robot je brankář a je řízen strategiemi automaticky bez možnosti tuto volbu změnit.

```
.Rule    1 o Name
.Mine    2,1  2,3
.Oppnt   5,1  5,3
.Ball    3,2
.Move    1,1  3,2
```

Výpis 6: Jedno pravidlo strategie

6 Ovládání robotů pomocí Bluetooth

Poslední ze tří hlavních oddílů výsledné aplikace je rozhraní pro ovládání robotů prostřednictvím rozhraní *Bluetooth*. Aplikace předaná předchůdcem byla napsána v jazyku *C++* a demonstrovala ovládání robota pomocí šipek a byla napojena na knihovnu pro ovládání robota sestaveného z *LEGO MINDSTORMS NXT*. Jak již bylo popsáno a vysvětleno výše, výsledný projekt byl zpracován v programovacím jazyku *C#* a proto sice došlo k analýze této aplikace, ale dále již v projektu použita nebyla. Na místo této aplikace tedy bylo potřeba použít jinou, určenou pro práci v programovacím jazyku *C#*.

Během výzkumu došlo na otestování dvou knihoven, které by mohly splňovat požadavky. Použitelnost knihovny byla rozhodnuta praktickou funkčností napojením na již existující verzi projektu. První testovanou knihovnou je *MindSqualls*[16], která obsahovala praktickou funkci pro ovládání více motorků zároveň. Dále došlo na testování knihovny *AForge.Robotics*[17].

6.1 MindSqualls knihovna

MindSqualls je knihovna, která byla napsána v jazyku *C#*, ale dle autora může být použita pro kterýkoli jazyk z rodiny *.NET*. *MindSqualls* se chlubí tím, že má v sobě zakomponovaný program *MotorControl*, který slouží k lepšímu ovládání motorů robota. Při ovládání dvou motorů tak, aby jel robot rovně, program zajistí, že se kolečka obou motorů otočí o přesně stejný počet otáček, takže se nestává, že by se robot stáčel do strany místo toho, aby jel rovně.

Ovládat robota lze potom tak, že je vytvořen objekt *McNxtMotorSync*, který představuje v programátorském kódu synchronizovaný pár motorů, a na tomto objektu následně dochází k volání metody *Run*, kde je mimo síly motoru a počtu otáček ještě potřeba zadat parametr *turnRatio*, který určí, kterým směrem robot zatočí.

Bohužel při testování těchto knihoven nedocházelo k požadovanému chování. Otáčení robota bylo spíše nahodilé a velmi často docházelo k tomu, že robot přestal reagovat na signály

posílané přes *Bluetooth*. Při ovládání jednotlivých motorů samostatně bez použití synchronizace robot nereagoval vůbec. Kvůli nízké spolehlivosti bylo proto vyvozeno, že funkčnost použité knihovny není uspokojující a přikročilo se tedy k vyhledání a otestování jiné knihovny pro ovládání robotů pomocí *Bluetooth*.

6.2 AForge.Robotics knihovna

Při hledání náhradní *Bluetooth* knihovny byla nalezena aplikace pro ovládání *LEGO MINDSTORMS NXT*, která využívala knihovny *AForge*, konkrétně knihovnu *AForge.Robotics*, která poskytuje podporu pro některé robotické sady včetně *LEGO MINDSTORMS NXT*. Knihovna *AForge* byla také napsána v programovacím jazyku *C#*, tudíž byla kompatibilní s vytvářenou aplikací.

Během testování knihovna *AForge.Robotics* nevykozavala žádné znatelné nedostatky nebo vedlejší účinky, pouze docházelo nepravidelně k situaci, kdy robot neprovede úkol kompletně, ale vykoná jen náznak zadaného pohybu. Při zaslání následujícího příkazu však robot již obvykle reaguje normálně. Tento menší nedostatek byl uvážen jako přijatelný a tak došlo k použití této knihovny ve výsledném projektu.

6.3 Rozhraní pro ovládání robotů

Nalezení správně fungující knihovny, ale neznamená, že aplikace bude schopna ihned správně ovládat roboty. Výstupem *modulu strategií* jsou totiž souřadnice, určující, kam se má robot posunout. Roboti však nemají žádnou informaci o své poloze a ovládání probíhá prostřednictvím příkazů, které říkají, o kolik se má otočit daný motor. Bylo proto třeba vytvořit rozhraní pro správný převod souřadnic na otáčky motoru.

Při výpočtu otáčení robota není otáčení robota o celý úhel mezi ním a požadovaným natočením ideálním řešením. Řekněme, že jde o robota, jehož cílové souřadnice vždy směřují na míč. Míč je však v neustálém pohybu, a když robot otočí požadovaným směrem, je už míč na jiné pozici a robot se musí otáčet znovu.

Otočení o velký úhel lze však nahradit otočením o několik dílčích úhlů. Problémem při vytváření rozhraní však je, že rychlost robota neodpovídá rychlosti procesu výpočtu nových souřadnic. Z toho důvodu je třeba počkat buďto určitý časový interval před posláním nových příkazů, což by mohlo být dostatečným činitelem pro snížení rozdílu mezi rychlostí robota a rychlostí aplikace, nebo je třeba posílat robotovi nové příkazy až ve chvíli, kdy se zastaví.

Řešením tohoto problému, které bylo použito ve výsledné aplikaci je možnost druhá. Neboli posílat nové příkazy robotovi, když už dokončil předchozí operaci. Negativním efektem tohoto způsobu je, že dochází k trhaným pohybům robota. Tento negativní efekt je však přijatelnější než kdyby robot nebyl schopen na požadované souřadnice dorazit vůbec.

Dalším problematickým krokem ve vývoji tohoto projektu bylo určení, kterým směrem se má daný robot otočit. Většina zdrojů pro výpočet úhlu mezi dvěma vektory totiž vrací absolutní hodnotu úhlu. Takový úhel nám tedy neřekne, kterým směrem od původního vektoru cílový vektor leží. Určuje pouze úhel, který svírají. *Modul strategií* implementuje vlastní objekt s názvem *Vector2D* pro ukládání a práci s vektory. Tento objekt byl u jiných operací užitečný, při výpočtu úhlu však disponuje stejným nedostatkem.

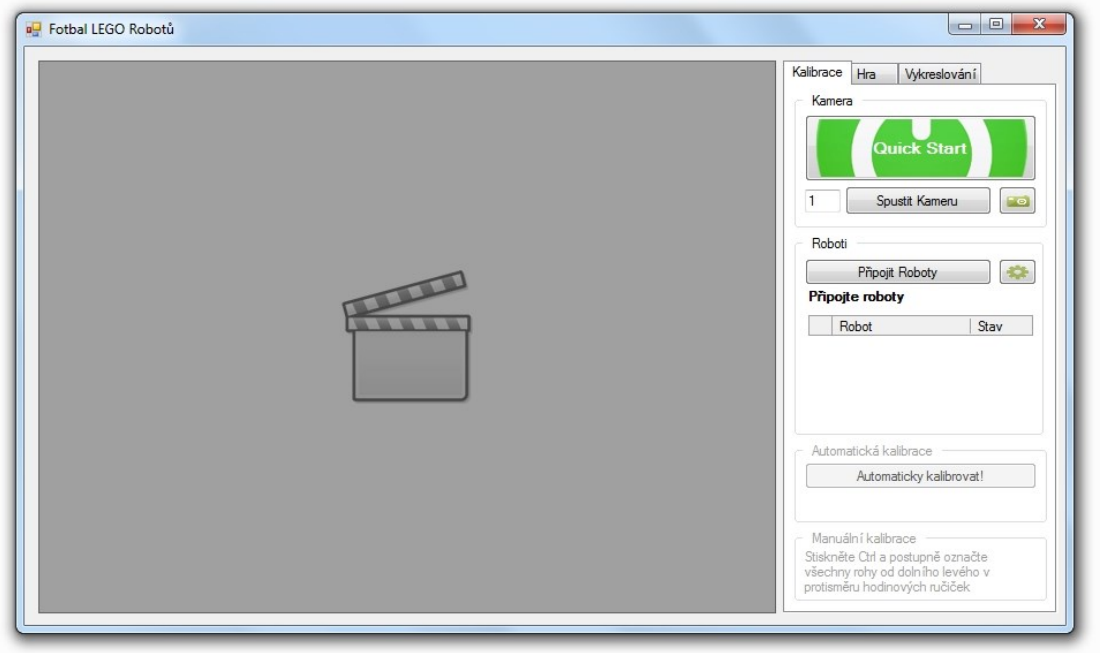
Řešením se ukázaly být knihovny vydané společností *Microsoft*, které přidávají navíc funkčnost základním jimi vydaným knihovnám. Tyto knihovny nejsou mezi základními referencemi, ale je třeba je vybrat jako reference z kategorie „*Extensions*“. Použitá knihovna *WindowsBase* rozšiřuje mimo jiné knihovnu *Systém.Windows* a obsahuje strukturu *Vector* včetně řady užitečných funkcí, které jsou s ní spojeny. V této struktuře je implementována metoda *AngleBetween* a na rozdíl od ostatních zmíněných funkcí pro výpočet úhlu, tato funkce vrací úhel i v záporných hodnotách, proto je dále snadné určit, zda mé robot zatočit vlevo nebo vpravo.

Metody implementovány v *Systém.Windows.Vector* a metody implementovány ve *Vector2D modulu strategií* se částečně překrývají a proto by bylo pro *modul strategií* přínosem, kdyby v něm byl *Systém.Windows.Vector* použit. Mohlo by tak dojít ke zlepšení funkcionality a

zvýšení efektivity této knihovny, protože zabudované knihovny obvykle dosahují většinou vyšší výkonnostní efektivity než uživatelsky vytvořené objekty.

7 Výsledná aplikace

Grafické rozhraní výsledné aplikace do jisté míry vychází z grafického rozhraní simulátoru fotbalu LEGO robotů. Tlačítka a panely se ale z větší části liší a tak výsledná aplikace dodržuje pouze stejné rozložení – nalevo zobrazení hry a napravo záložky s ovládacími panely.

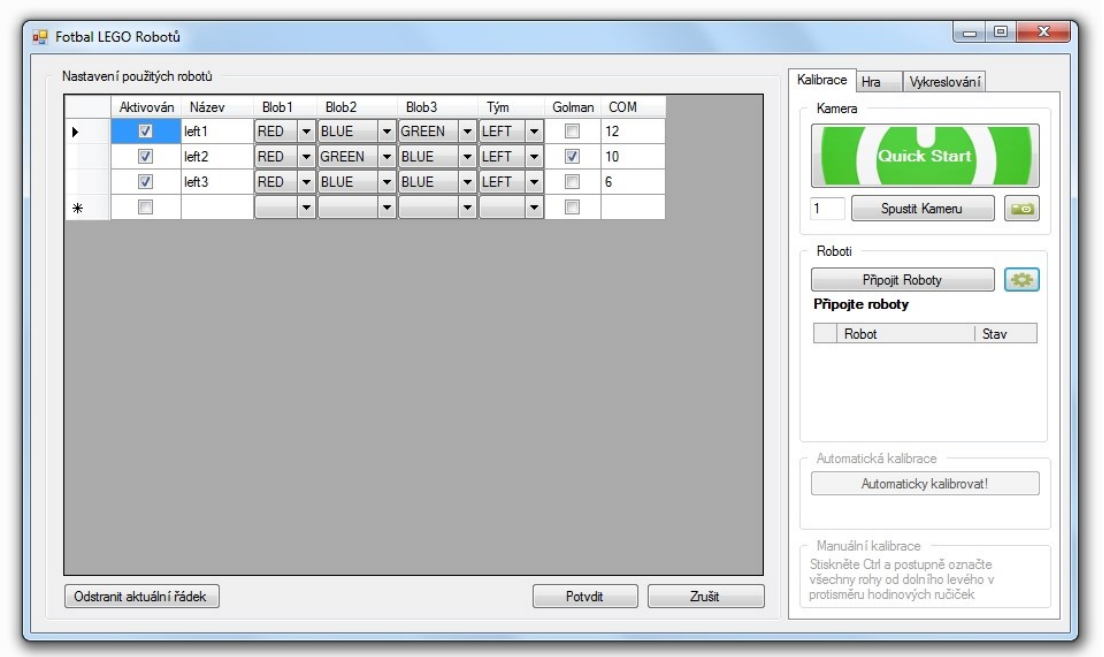


Obrázek 20: Vzhled aplikace ihned po spuštění, Zdroj vlastní

První záložka nese název *Kalibrace*. Obsahuje čtyři panely. První slouží pro připojení kamery. Obsahuje nejprve tlačítko *Quick Start*. Dále obsahuje tlačítko pro připojení nebo odpojení kamery, které připojí kameru podle indexu uvedeného v textovém poli. A tlačítko pro uložení aktuálního zobrazeného snímku.

Další záložka se jmenuje *Roboti*. Umožňuje nastavit informace o robotech, kteří budou použiti ve hře, stejně jako tyto informace měnit. Aby aplikace neobsahovala mnoho oken,

které nejsou uživatelsky přívětivé, bylo nastavování robotů vloženo jako panel, který se po kliknutí na tlačítko s ikonou nastavení zobrazí uvnitř aktuálního okna. Po stisku tlačítka *Připojit roboty* se aplikace pokusí prostřednictvím *Bluetooth* připojit ke všem přednastaveným robotům a níže vypíše, zda byla operace úspěšná.



Obrázek 21: Vzhled aplikace při úpravě nastavení jednotlivých robotů, Zdroj vlastní

Následují panely pro *automatickou* a *manuální kalibraci* hřiště. Aplikace je navržena tak, že je třeba projít všemi třemi kroky, než může uživatel přejít k nastavování možností v panelech na další záložce. Aby tato operace nebyla zdlouhavá je v panelu umístěno již zmíněné tlačítko *Quick Start*. Toto tlačítko slouží k provedení všech tří operací jediným kliknutím a zobrazí uživateli rovnou dotaz, zda byla automatická kalibrace úspěšná. Pokud však automatická kalibrace hřiště úspěšná nebyla, je třeba provést kalibraci manuální.

Následující panel pod názvem *Hra* slouží k nastavení strategií jednotlivých týmů a spuštění samotné hry *fotbalu LEGO robotů.* Jsou zde také informace týkající se informací o hře,

především skóre hry a časomíra. Navíc jsou zde pro uživatele zobrazeny časy vykonávání jednotlivých dílčích procesů aplikace v milisekundách.

Poslední panel je užitečný při zjišťování, proč se aplikace nechová, jak by měla. Umožňuje totiž zobrazit nebo skrýt vytažení nalezených identifikačních štítků, stejně tak identifikaci celých robotů a pozice, na kterou by se měly přesunout dle výpočtů *modulu strategií*.

8 Testování

V následující kapitole se zaměřím na experimenty pro otestování, zda jednotlivé části fungují správně samostatně i jako celek. Také se pokusíme odhalit nedostatky, které by aplikace mohla mít.

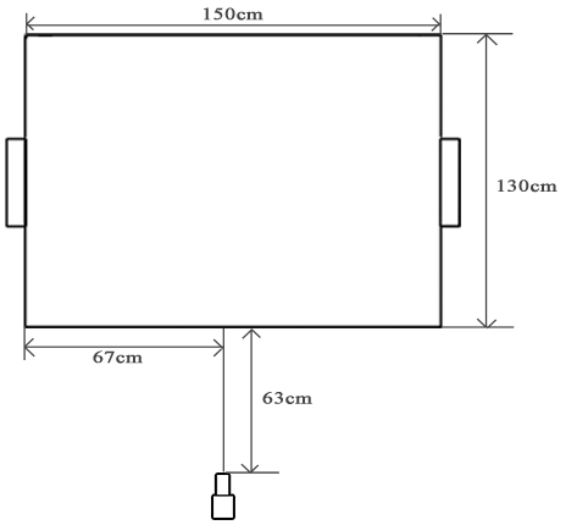
První test je zaměřen pouze na analýzu obrazu, a jak si poradí s různým umístěním kamery. Druhý test ve spolupráci a analýzou obrazu vyzkouší rozhraní pro ovládání *Bluetooth*. A třetí test se zaměří na otestování, zda se roboti opravdu chovají podle zabudovaného *modulu strategií*.

7.1 Testování analýzy obrazu

Při testování analýzy jsem se zaměřil na umístění kamery. Použitou kameru nebylo možné umístit na rám kolem hřiště, protože kvůli pevné ohniskové vzdálenosti nebylo v obraze vidět celé hřiště. Proto byla kamera umístěna na stativ vedle hřiště. Byly provedeny čtyři testy, pokaždé s jiným nastavením kamery vůči hřišti. Hřiště bylo po celou dobu testování ve stejné pozici na zemi a značení robotů tak bylo pouze ve výšce několika centimetrů od podlahy místnosti.

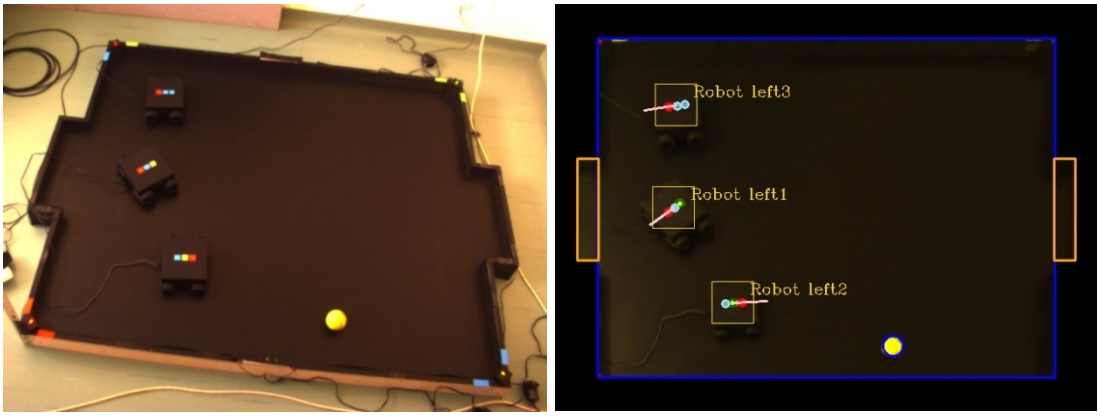
Během těchto testů byl kladen důraz na to, zda byla aplikace po manuální kalibraci obrazu schopna identifikovat správně jednotlivé identifikační štítky a tudíž i samotné roboty.

7.1.1 První pozice kamery



Obrázek 22: umístění kamery během prvního snímání, Zdroj vlastní

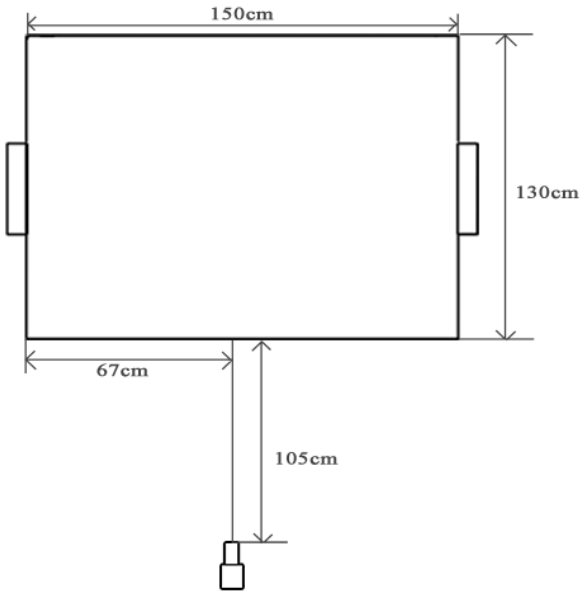
V prvním testu byla kamera umístěna do pozice, která je vyobrazena v nákresu výše. Identifikační značení robotů Kamera byla v tuto chvíli ve výšce 190cm nad podlahou místnosti. Následující fotografie zachycují vzhled hřiště v zachyceném snímku před a po kalibraci hřiště.



Obrázek 23 a 24: Vzhled hřiště před a po kalibraci z první pozice, Zdroj vlastní

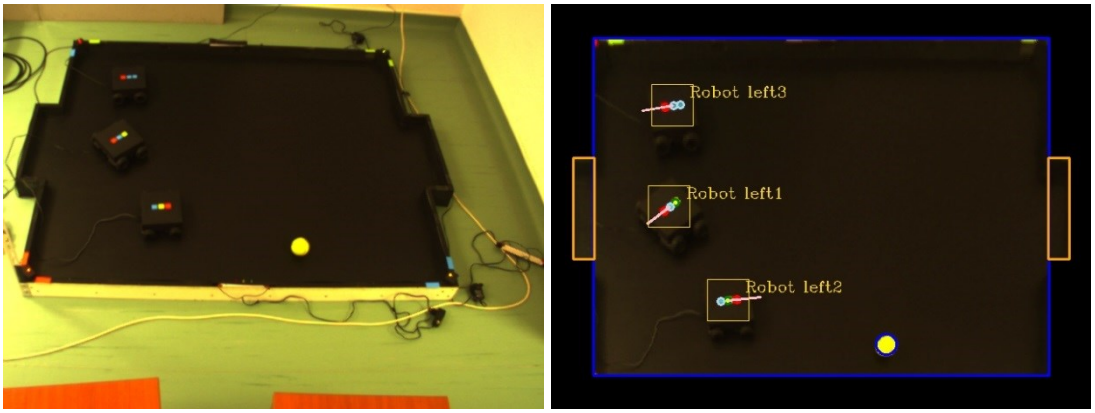
Ze snímku zachyceného kamerou (obrázek 23) vidíme, že kolem hřiště není v obraze mnoho místa, ale hřiště z velké části snímek vyplňuje. Dále je vidět, že dochází k menší až střední deformaci tvaru hřiště. Na snímku po kalibraci hřiště (obrázek 24) je však vidět, že deformace tvaru hřiště byla v takovém rozsahu, který aplikace dokáže napravit a úspěšně identifikovat jednotlivé roboty.

7.1.2 Druhá pozice kamery



Obrázek 25: Pozice kamery během druhého a třetího testu, Zdroj vlastní

Během druhého testu byla kamera umístěna k hřišti ze stejného pohledu jako v prvním testu. Byla však zvětšena vzdálenost vůči hřišti při zachování výšky kamery vůči zemi 190cm, což znamenalo, že hřiště v obraze vyplňovalo méně prostoru a úhel z jakého kamera na hřiště shlížela, se změnil.

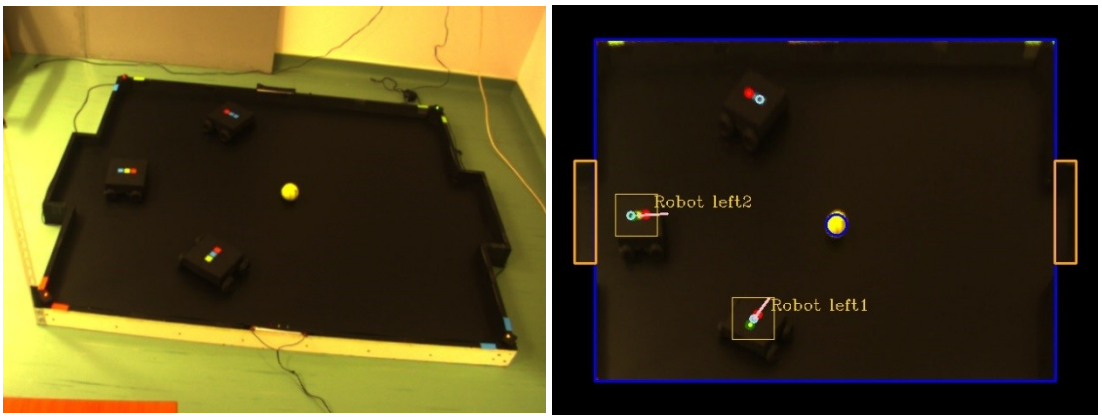


Obrázek 26 a 27: Vzhled hřiště před a po kalibraci z druhé pozice, Zdroj vlastní

Ze snímku zachyceného kamerou z druhé pozice je vidět, že hřiště již v obraze zabírá méně místa. Deformace tvaru hřiště je navíc větší, než během předchozího snímání. Aplikace však byla stále schopna chyby obrazu napravit a podařilo se jí identifikovat všechny roboty i míč.

7.1.3 Třetí pozice kamery

Během třetího snímání byla kamera ponechána ve stejné pozici jako během snímání druhého. Výška kamery však byla v tomto případě snížena na 165cm. Z této pozice deformace obrazu opět vzrostla.

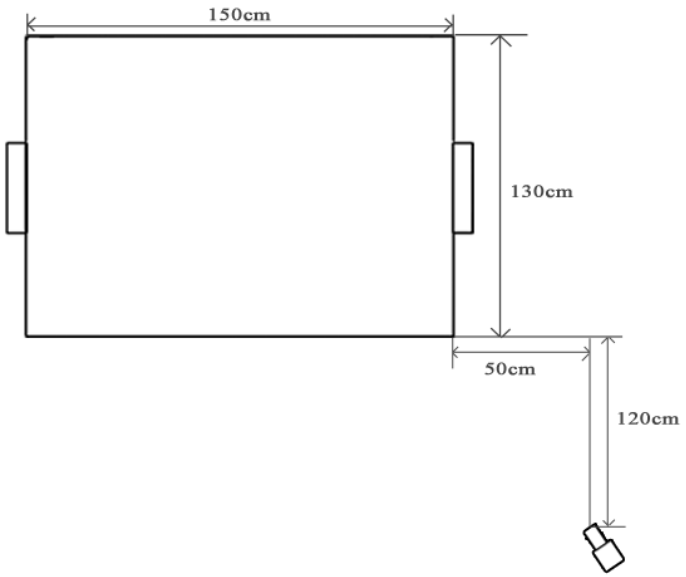


Obrázek 28 a 29: Vzhled hřiště před a po kalibraci z třetí pozice, Zdroj vlastní

Z tohoto úhlu pohledu již *analýza obrazu* vykazovala obtíže s detekováním hráčů v některých pozicích jak je vidět na obrázku 29. Na míči je již také vidět známky deformace, kdy se kulatý tvar míče změnil na eliptický.

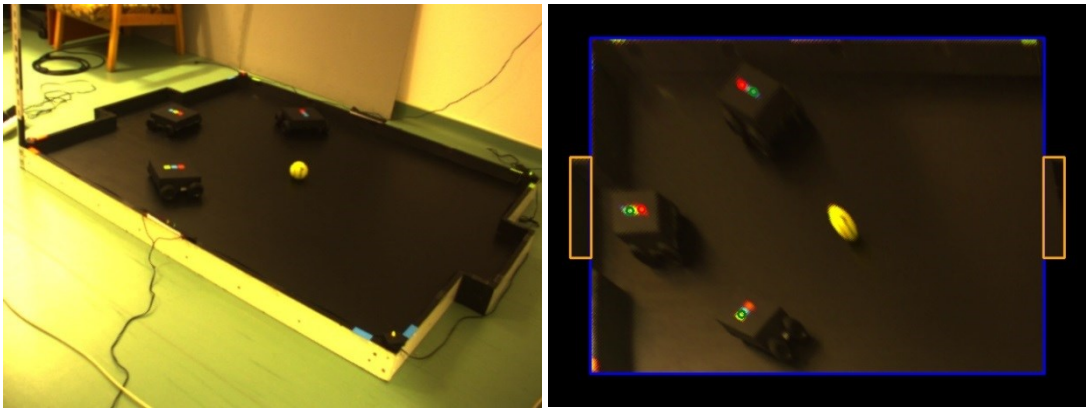
Problém s detekcí hráče však nebyl v tom, že by identifikační štítek již neměl původní tvar, ale že identifikační štítky byly umístěny příliš blízko a pokud byly dva štítky stejné barvy na jednom robotovi vedle sebe, tak se v obraze nepodařilo rozeznat předěl mezi těmito štítky a aby byly tak identifikovány pouze jako jeden štítek, což zapříčinilo selhání identifikace tohoto robota.

7.1.4 Čtvrtá pozice kamery



Obrázek 30: Pozice kamery během čtvrtého testu, Zdroj vlastní

Čtvrté a poslední umístění kamery představovalo spíše extrémní situaci. Kamera byla v tomto případě umístěna 105cm nad podlahou místnosti. Rotace hřiště v obraze a výška kamery v tomto případě již způsobovaly silnou deformaci obrazu.



Obrázek 31 a 32: Vzhled hřiště před a po kalibraci ze čtvrté pozice, Zdroj vlastní

Ve snímku ze čtvrtého snímání je vidět, že je hřiště rozprostřeno po celé šířce obrazu. Vzhledem rotaci a deformaci tvaru hřiště však vyplňuje spíše menší část snímku. Snímek po kalibraci obrazu ukazuje, že aplikace analýzy obrazu si s tímto stupněm deformace již nedokázala poradit, obzvláště proto, že se smýval přechod mezi jednotlivými identifikačními štítky.

7.1.5 Vyhodnocení

První a druhé snímání ukázalo, že čím je kamera umístěna z vyššího pohledu vůči hřišti, tím je to vytvořenou aplikaci výhodnější a narůstá úspěšnost identifikace. V těchto dvou prvních snímáních neměla aplikace žádný problém s identifikací robotů.

Třetí snímání však ukázalo, že *analýza obrazu* již začíná mít problémy s rostoucí vzdáleností a zhoršujícím se úhlem, což také znamená horší rozlišení obrazu v části snímku, kde je vyobrazeno hřiště. Při praktickém použití se však nepředpokládá, že by byla kamery umístěna ve větší vzdálenosti, než je potřeba. Čtvrtý test pouze potvrdil tyto závěry.

Třetí snímání však bylo velkým přínosem, protože odhalilo, že jsou jednotlivé snímky příliš blízko jeden druhému, což způsobuje výše uvedené chyby v identifikaci hráčů. Proto bylo identifikační značení předěláno způsobem, který následně zvýšil úspěšnost správné

identifikace tím, že byl zvětšen rozestup mezi jednotlivými štítky z 5mm na 15mm. Tento upravený způsob rozložení štítků tedy nahradil původní způsob a je popsán v kapitole 3.2.4.

7.2 Testování Bluetooth rozhraní

Bluetooth rozhraní bylo otestováno za pomocí pouze jednoho robota. Cílem tohoto testu nebylo pouze ověřit, zda bude robot opravdu zatáčet doleva nebo doprava. *Bluetooth rozhraní* totiž provádí přepočet souřadnic a tak bylo cílem ověřit, zda robot opravdu dojede na souřadnice, které jsou *Bluetooth rozhraní* předány. Do aplikace byly vloženy tři body, a robotovým úkolem bylo jeden po druhém postupně projet.

Protože není možné, aby se robot trefil na bod s přesností jednoho pixelu, byla nastavena tolerance na 20 pixelů, což je při aktuálním nastavení kamery a robotů méně než polovina hrany robota. Tato tolerance vychází z předpokladu, že pokud by místo testovacího bodu byl cílem míč, robot by se do něj měl alespoň trefit.

Tento pokus byl proveden na všech třech dostupných robotech. Každý robot byl umístěn na různou startovací pozici, aby byla vyvrácena možnost, že je robot schopen se správně otočit pouze z určitého umístění. První robot byl umístěn na střed hřiště a z pohledu kamery mířil směrem nahoru. Druhý robot byl umístěn do levého dolního rohu a byl natočený směrem doleva. Třetí robot zaujímal pozici v pravém horním rohu a byl otočen směrem doprava.

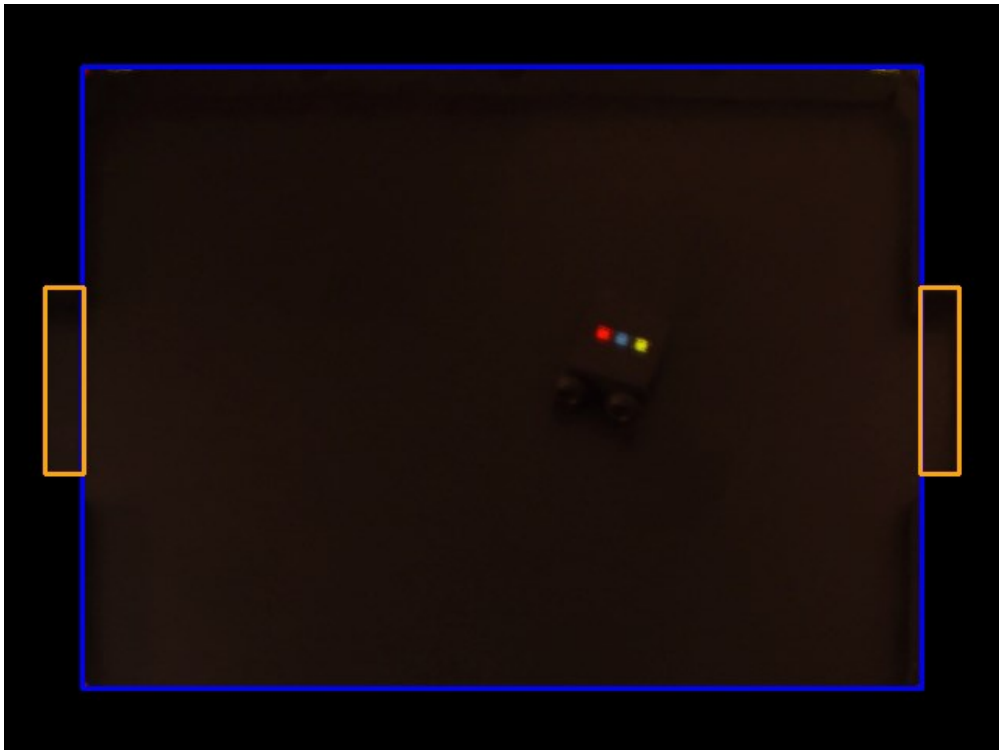
Následně byl spuštěn test. V tomto testu se podařilo všem robotům úspěšně projet všechny stanovené body. Po tomto testu byla provedena změna dvou nastavení *Bluetooth rozhraní*. Během testu se ukázalo, že konstrukce robota způsobuje vedlejší efekt, kdy se robot neotáčí kolem svého středu, ale ve skutečnosti během otáčení vždy robot popojede dozadu. Při otáčení se kolečka na jedné straně točí vpřed a na druhé straně vzad, proto bylo nastavení upraveno tak, aby kolečka, která se točí vzad, jela pouze na 90% výkonu. Druhým nastavením, které se dočkalo změny, byla tolerance úhlu, pro jízdu vpřed. U prvního pokusu jel robot vpřed vždy, když úhel mezi jeho rotací a požadovanou rotací byl menší nebo roven 15° na každou stranu. Tato tolerance byla u druhého pokusu změněna na 12°. Následující tabulka ukazuje časy průjezdů robotů danými body v obou testovacích jízdách.

Robot	Startovní pozice	Pokus	Bod1 [s]	Bod2 [s]	Bod3 [s]
Robot1	Levý dolní roh, Otočený doleva	Pokus1	7	16	26
		Pokus2	6	12	20
Robot2	Střed hřiště, Otočený nahoru	Pokus1	18	26	35
		Pokus2	21	28	38
Robot3	Pravý horní roh, Otočený doprava	Pokus1	11	20	30
		Pokus2	12	19	27

Tabulka 1: Startovní pozice a časy průjezdů robotů jednotlivými body, Zdroj vlastní

Po změně nastavení došlo k tomu, že druhý robot dojel k prvnímu bodu o tři sekundy později a třetí robot o jednu sekundu později. Celkové rozdíly v časech průjezdů mezi prvním a druhým pokusem však ukazují, že druhé použité nastavení dosahuje lepších výsledků a tak bylo ponecháno.

Jako přídavek k tomuto testu se objevila možnost otestovat, jak vypadá na zachyceném snímku robot, který se pohybuje. Při porovnání vzhledu robota z obrázku 33 s robotem, který se nepohybuje, dojdeme k závěru, že jsou identifikační štítky v obou případech jemně rozmazané, přesto jsou stále ucelené a jeden neovlivňuje negativně druhý, což platí i o snímku robota, který je v pohybu. Toto snímání také prokázalo, že výměna kamery byla dobrým krokem, protože u předchozí kamery byl pohybující se robot neidentifikovatelný. Následující obrázek ukazuje, jak pohybující se robot vypadá v aplikaci, když není zapnuto zvýrazňování štítků a robotů.



Obrázek 33: Štítky jsou stále znatelně odděleny i u pohybujícího se robota,
Zdroj vlastní

7.2.1 Vyhodnocení

Tento test byl do jisté míry úspěšný a ukazuje, že *Bluetooth* rozhraní je dostatečně schopno navést robota na zvolené místo na hřišti tím, že samo vyhodnotí, zda by robot měl jet rovně nebo se otočit doleva či doprava a následně pošle robotovi odpovídající příkaz. Tento test také odhalil, že testovaní roboti nejezdí naprosto rovně, ale při jízdě vpřed mají tendenci stáčet se na jednu nebo druhou stranu. Zabudované motorky totiž neotočí kolečka vždy o stejný úhel. Tento nedostatek však není natolik výrazný, aby měl výraznější vliv na ovládání robotů.

7.3 Testování modulu strategií

Klasický *fotbal robotů* obvykle spočívá ve hře s pěti hráči v každém týmu. Testování reálné hry 5 proti 5 nebylo možné z důvodu nedostatku LEGO robotů. Celkově jsou momentálně dostupní tři LEGO roboti, což nestačí ani pro sestavení jednoho takového týmu. Ve *fotbalu LEGO robotů* však není nutné, aby tým měl pět hráčů.

Z tohoto důvodu jsem umístil všechny tři hráče do hřiště jako členy jednoho týmu, zatímco druhý tým zůstane prázdný. Hru tedy hrají týmy o třech hráčích, zatímco v jednom týmu jsou hráči skuteční, v druhém jsou pouze proto, aby *modul strategií* měl nezbytné údaje pro hru. Experiment tedy spočívá v otestování, zda strategie dokáže vyhodnotit vstupní data správným způsobem. Pokud se roboti pohnou ze svých pozic na pozice definované v souboru strategií, pak *modul strategií* vyhodnotil data správně.

Nejprve bylo potřeba vytvořit soubor s pravidly pro *modul strategií*. Jak bylo popsáno výše, je to jednoduchý textový soubor, do kterého jsou zapsána pravidla v textovém formátu. Pravidla použita pro účely tohoto textu jsou vypsána níže.

```
.Rule    1 o Name
.Mine    3,2  3,4
.Oppnt   7,2  7,4
.Ball    5,3
.Move    5,3  3,3

.Rule    2 o Name
.Mine    5,3  3,3
.Oppnt   6,3  6,5
.Ball    5,4
.Move    4,4  3,2

.Rule    3 o Name
.Mine    4,4  3,2
.Oppnt   4,3  3,5
.Ball    2,3
.Move    2,4  2,2

.Rule    4 o Name
.Mine    3,4  5,2
.Oppnt   6,1  3,5
.Ball    7,3
.Move    7,4  4,3)
```

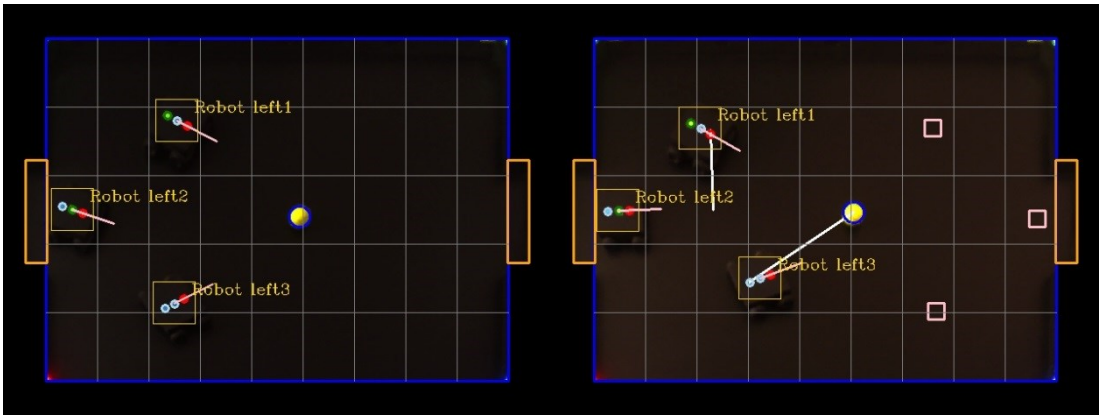
```
.Rule    5 o Name
.Mine    4,2  7,3
.Oppnt   9,2  9,4
.Ball    8,3
.Move    8,3  4,3

.Rule    6 o Name
.Mine    5,3  8,3
.Oppnt   7,5  8,2
.Ball    9,3
.Move    9,3  5,3

.Rule    7 o Name
.Mine    3,4  6,1
.Oppnt   7,5  8,2
.Ball    6,1
.Move    3,4  5,1

.Rule    8 o Name
.Mine    3,4  5,1
.Oppnt   7,5  8,2
.Ball    6,1
.Move    3,4  6,1
```

Výpis 7: Pravidla strategie pro testovací účely

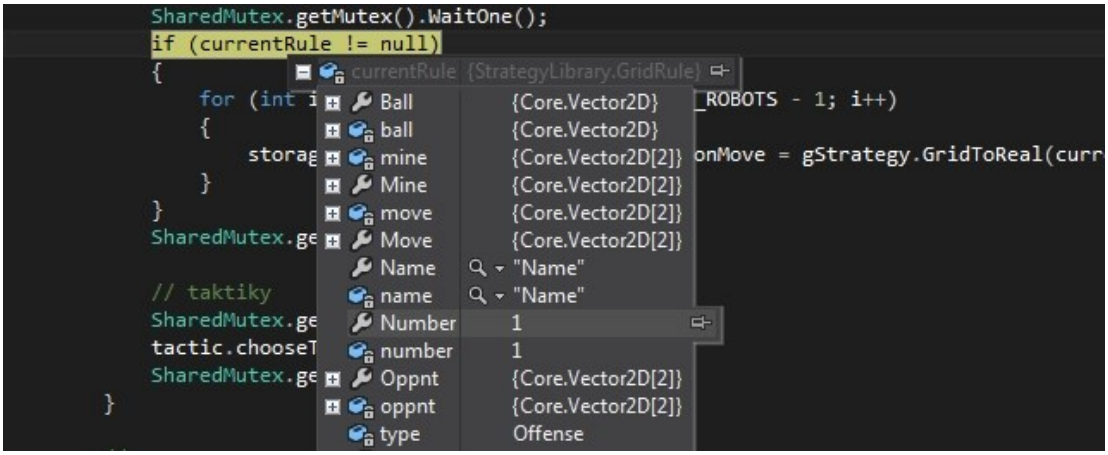
Během testování bude využíváno obrázků, které budou obsahovat grafické značení jednotlivých pozic. *Žlutý čtverec* znázorňuje robota, *růžové čtverce* znázorňují údajné pozice protihráčů, které byly předány modulu *strategií*, aby mohl provádět požadované výpočty. Dále od robotů znázorněnými žlutými čtverci může vést bílá čára, která znázorňuje kam se má podle výpočtu daný robot přesunout.

7.4.1 První situace - start

Nejprve byli roboti umístěni do postavení, které se považovalo za normální startovní postavení robotů na hřišti. Toto postavení odpovídá postavení, které je popsáno v prvním pravidlu použitého souboru strategií.



Obrázek 34 a 35: Startovní pozice a pohyb hráčů v první situaci, Zdroj vlastní

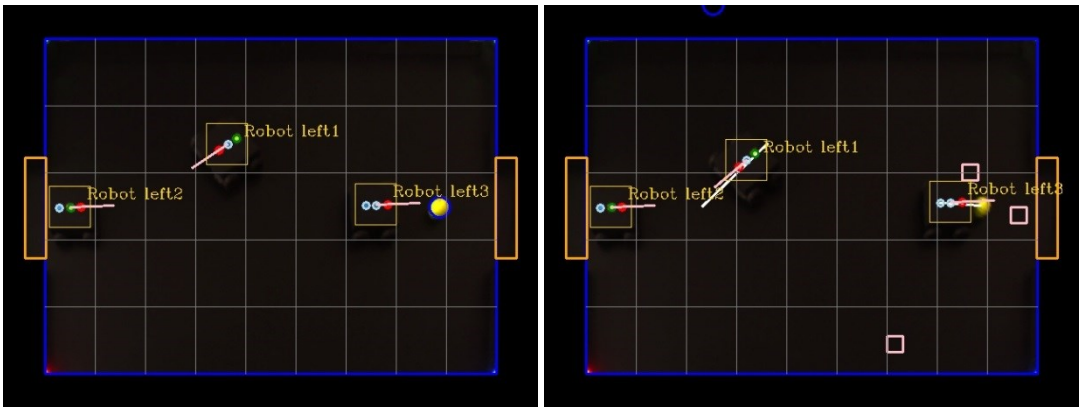


Obrázek 36: Modul strategií zvolil první pravidlo, Zdroj vlastní

Na obrázku 31 jsou roboti ve stavu, před započnutím hry, kdy ještě nedostaly žádné příkazy. Počáteční pozice přímo odpovídá prvnímu pravidlu a na obrázku 33 je vidět, že toto pravidlo také bylo strategií vybráno. Na obrázku 32 již jsou roboti v pohybu. Bílé čáry zobrazují cíl daného robota. Cíle jednotlivých robotů odpovídají cílům nastaveným v prvním pravidle souboru pravidel a roboti se daným směrem skutečně vydali.

7.4.2 Druhá situace - útok

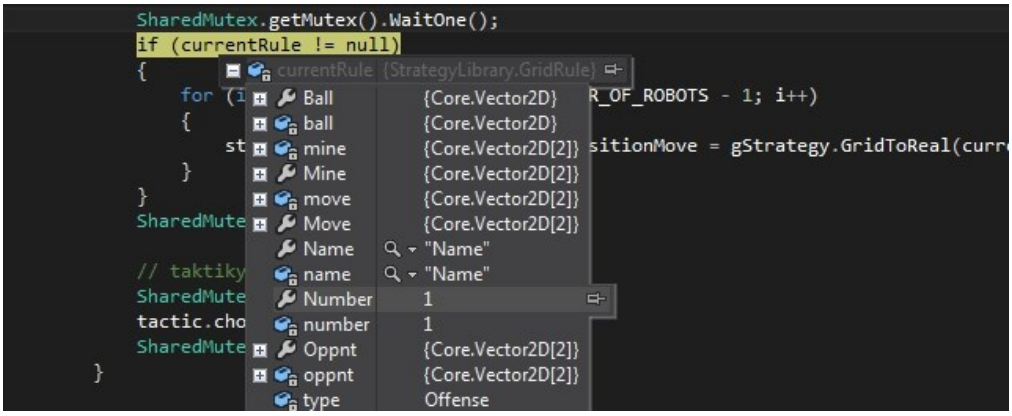
V druhé situaci bude vyzkoušena schopnost robotů útočit. Roboti byli umístěni do pozice vyobrazené na obrázku 34. Obrázek 35, který následuje, ukazuje, že jeden robot se stáhnul do obranné pozice, zatímco druhý se vydal vstříc míči, aby jej tlačil směrem do brány.



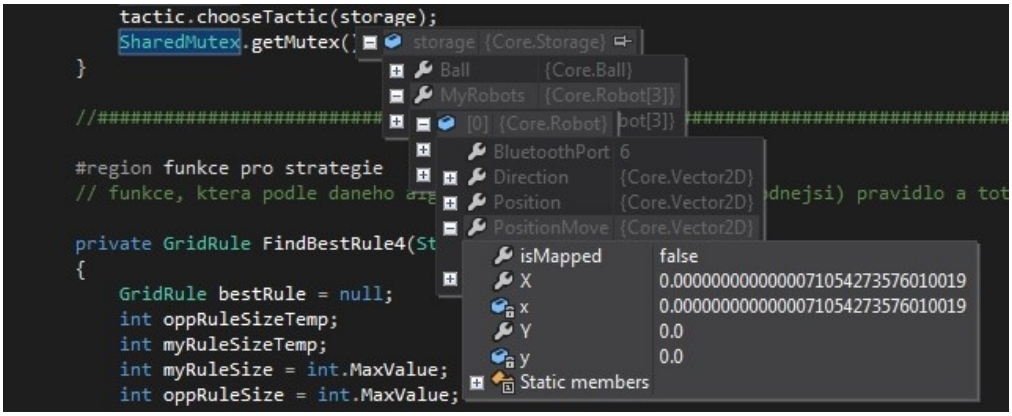
Obrázek 37 a 38: Startovní pozice a pohyb hráčů v druhé situaci, Zdroj vlastní

Tato reakce však není přesně tou, jaká byla původně očekávána. Na počátku stáli roboti na souřadnicích 4,2 a 7,3, které přesně odpovídají pravidlu číslo 6. Robot *left3* se skutečně pohnul za míčem a robot *left1* do defenzivního postavení, přesunul se však na jinou pozici, než určovalo pravidlo 6. Na obrázku 39 můžeme vidět, že *modul strategií* této situaci přiřadil pravidlo číslo 1 namísto očekávaného pravidla 6.

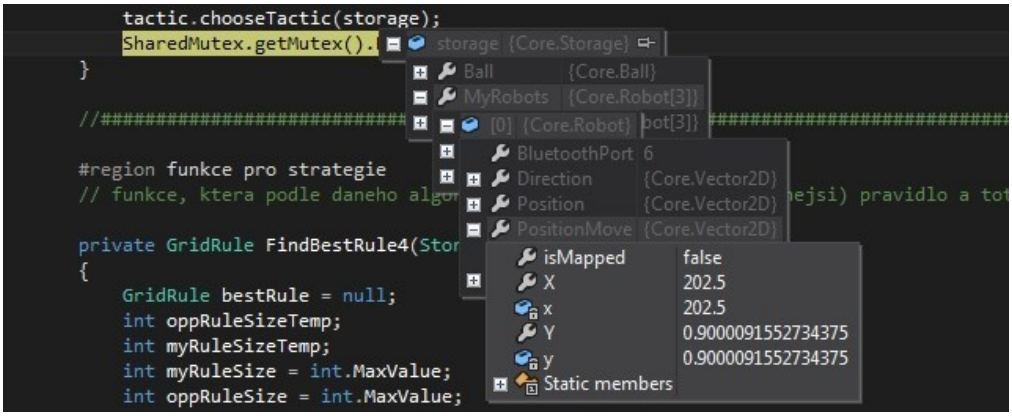
Útočící robot by se v takové situaci měl stáhnout do obranné pozice. Obrázek 40 opravdu ukazuje, že byl nejprve jeho cíl nastaven na pozici 0,0, neboli střed hřiště. Po výběru nejvhodnějšího pravidla však ještě probíhá výběr taktik. Během této operace byl robot poslán na míč, jak je vidět z obrázku 41.



Obrázek 39: Modul strategií přiřadil dané situaci pravidlo číslo 1, Zdroj vlastní



Obrázek 40: Před provedením taktik byl robot poslán na střed hřiště, Zdroj vlastní

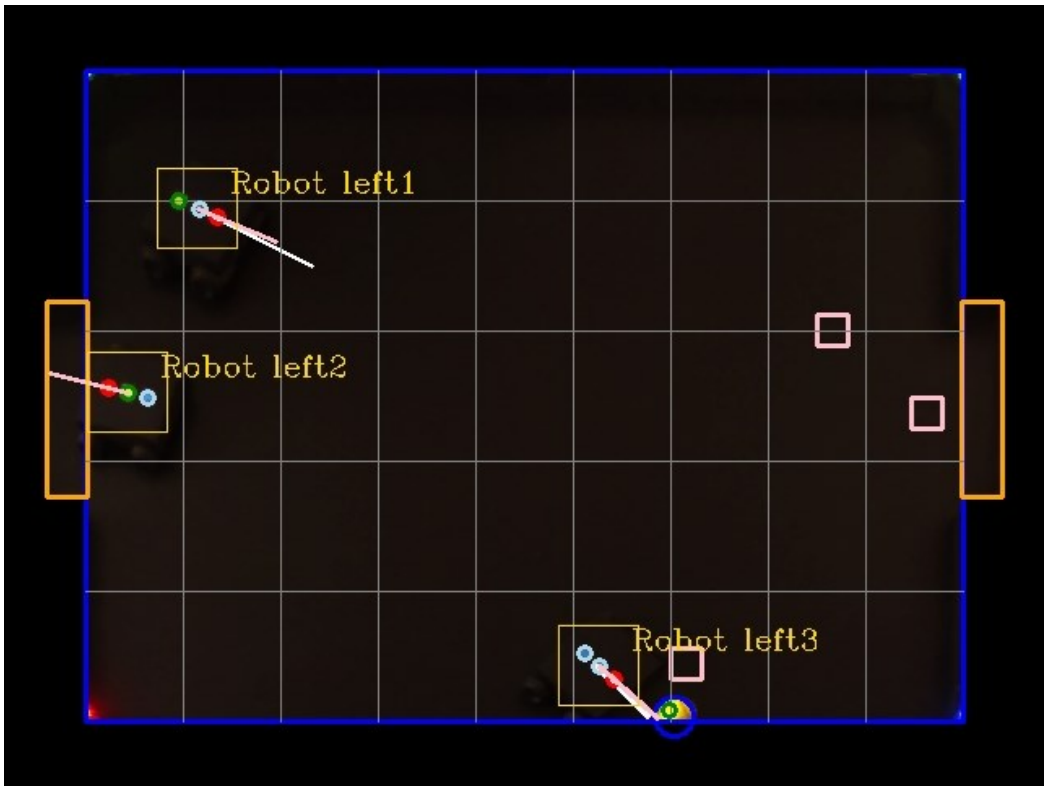


Obrázek 41: Po provedení taktik byl robot poslán na míč, Zdroj vlastní

Dalším odkrokováním programu bylo dokázáno, že *modul strategií* opravdu dokáže vybrat i jiné pravidlo než to první. Z výsledků tohoto testu se sice projevilo se, že strategie ne vždy vybere to nejlepší pravidlo, přesto však *modul strategií* obsahuje výběr taktik, které dokáže tuto chybu částečně napravit.

7.4.3 Třetí situace – u mantinelu

Následující situace slouží jako ukázka konstrukčních nedostatků robotů a míče. Původní roboti pro *fotbal robotů* byli navrženi tak, aby byli malí, rychlí. Navíc velmi důležitým konstrikčním prvkem bylo to, že kolečka byla schovaná uvnitř těla robota. Díky této konstrukci je kolize s jinými hráči nebo mantinely příliš neovlivňovaly. To se však nedá říci o *LEGO robotech*. Míč určený pro původní *fotbal robotů* se navíc po hřišti pohyboval velmi rychle a snadno se odrážel.



Obrázek 42: Robot a míč v blízkosti mantinelu, Zdroj vlastní

Obrázek 39 ukazuje situaci, kdy je míč blízko mantinelu, protože neměl dostatečnou razanci a odrazivost, aby pokračoval v pohybu více do středu hřiště. Robot *left3* přijel k míči, nebyl však schopen se k němu dostat, protože jeho pravé kolečko zavadilo o mantinel hřiště a znemožňuje mu tak otočit se směrem k míči. Omezit pozice, na které se může robot přesunout tak, aby robot nikdy nenarazil do mantinelu, by nebylo dostatečným řešením, protože by pak byl robot schopen pouze dotlačit míč do rohu hřiště, odkud by jej již nebyl schopen dostat.

7.4.4 Vyhodnocení

Na hřišti během testování byli přítomni pouze roboti jednoho týmu, proto se můžeme pouze domnívat, jak by hra vypadala, kdyby hráli i roboti týmu druhého. Toto testování ale ukázalo, že *modul strategií* úspěšně přijímá data předaná *analýzou obrazu* a dokáže je zpracovat a předat *Bluetooth rozhraní*. Ukázalo se ale také, že je někdy vybráno pravidlo, které nejvíce odpovídá situaci. Dále se ukázalo, že sestavení robotů ze stavebnice *LEGA* není ideálním řešením, protože tato konstrukce přináší řadu vedlejších účinků. Roboti tak jsou schopni hrát *fotbal LEGO robotů*, ale stávalo se, že se roboti z konstrukčních a strategických důvodů dostali do situace, odkud pro ně bylo náročné pohnout se dále. Konstrukce robotů a modul strategií jsou proto doporučeny k dalšímu vývoji.

8 Závěr

Na začátku byly tři samostatné aplikace. Tento projekt úspěšně spojil dohromady *LEGO roboty*, *analýzu obrazu*, *modul strategií* a *Bluetooth rozhraní*, přičemž některé z uvedených částí byly z části nebo kompletně přepracovány.

Konstrukce robotů neumožňovala velké změny, proto byla zachována. V průběhu praktického používání však bylo odhaleno, že původní značení pomocí *LED diod* nebylo spolehlivé a bylo nahrazeno značením pomocí barevných štítků, které přinesly zvýšení úspěšnosti detekce a identifikace robotů. Část aplikace provádějící analýzu obrazu využila část algoritmů z aplikace předchůdce. Z velké většiny však používala algoritmy jiné. Část aplikace vyhodnocující strategie byla upravena pro dynamický počet robotů a zabudována do vyvíjené aplikace. Část aplikace pro ovládání robotů nebyla v původní verzi dostačující a tak byla napsána celá od začátku. Grafické rozhraní aplikace se skutečně chovala jako jednotný celek, který je možno ovládat z relativně jednoduchého grafického rozhraní.

Testování výsledné aplikace ukázalo, že kamera nemusí být vždy umístěna přímo nad středem hřiště ani nemusí být vždy na stejném místě. Tento způsob snímání hřiště tak přináší aplikaci další dynamický prvek. Dále testování ukázalo, že je aplikace schopna přesně navést roboty na určené místo. Ukázalo se však také, že pro samotnou hru *fotbalu LEGO robotů* není použitá konstrukce robotů ideální a přináší několik nedostatků.

Do budoucna je doporučeno dále některé složky tohoto projektu rozvíjet. Pro značení robotů je například možno použít infračervené *LED diody* nebo bezdrátový signál, jak bylo doporučeno výše.

Reference

[1] History of robots (Wikipedie) [online]
http://en.wikipedia.org/wiki/History_of_robots [cit. 2013-5-4]

[2] Outline of robotics (Wikipedie) [online]
http://en.wikipedia.org/wiki/Outline_of_robotics [cit. 2013-5-4]

[3] ASIMO by Honda | The World's Most Advanced Humanoid Robot [online]
http://asimo.honda.com/ [cit. 2013-5-4]

[4] A BriefHistoryofRoboCup [online]
http://www.robocup.org/about-robocup/a-brief-history-of-robocup/ [cit. 2013-5-4]

[5] PASTRŇÁK, J. *MicrosoftRobotics Studio – Programování robotů*. Ostrava, 2008. 60 s.
Diplomová práce, Katedra informatiky, VŠB – Technická univerzita Ostrava.

[6] NEUSTUPA, Z. *Fotbal Lego Robotů*. Ostrava, 2012. 52 s.
Diplomová práce, Katedra informatiky, VŠB – Technická univerzita Ostrava.
Vedoucí diplomové práce RNDr. Eliška Ochodková, Ph.D.

[7] qt-opencv-multithreaded [online]
http://code.google.com/p/qt-opencv-multithreaded/ [cit. 2013-7-18]

[8] Open Source Initiative - The MIT License (MIT) [online]
http://opensource.org/licenses/mit-license.php [cit. 2013-7-14]

[9] www.philohome.com [online]
http://www.philohome.com/nxtmotor/nxtmotor.htm [cit. 2013-7-14]

[10] LEGO MINDSTORMS® EV3 FrequentlyAskedQuestions [online]
http://mindstorms.lego.com/en-us/News/ReadMore/Default.aspx?id=476781
[cit. 2013-7-14]

[11] OpenCv – About [online]
http://opencv.org/about.html [cit. 2013-7-17]

[12] OpenCVWiki [online]
http://opencv.willowgarage.com/wiki/ [cit. 2013-7-17]

[13] Emgu CV - WorkingwithImages [online]
http://www.emgu.com/wiki/index.php/Working_with_Images#Creating_Image
[cit. 2013-7-17]

[14] Wikipedie – HSV [online]

http://cs.wikipedia.org/wiki/HSV [cit. 2013-7-17]

[15] Andrew Harvey's Blog - COMP3421 – Lec 1 – Colour [online]
http://andrewharvey4.wordpress.com/2009/08/22/comp3421-lec-1-colour/
[cit. 2013-7-23]

[16] MindSqualls [online]
http://www.mindsqualls.net/ [cit. 2013-7-18]

[17] AForge.NET [online]
http://code.google.com/p/aforge/ [cit. 2013-7-18]

[18] LEGO MINDSTORMS NXT [online]
http://mindstorms.lego.com/en-gb/history/default.aspx [cit. 2013-5-4]